ENCODING 3D CONTEXTUAL INFORMATION FOR DYNAMIC SCENE UNDERSTANDING

A Dissertation
Presented to
The Academic Faculty

By

Steven Hickson

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Georgia Institute of Technology

Georgia Institute of Technology

May 2020

ENCODING 3D CONTEXTUAL INFORMATION FOR DYNAMIC SCENE UNDERSTANDING

Approved by:

Dr. Irfan Essa, Advisor
School of Interactive Computing
Georgia Institute of Technology

Dr. Frank Dellaert
School of Interactive Computing
Georgia Institute of Technology

Dr. Judy Hoffman
School of Interactive Computing
Georgia Institute of Technology

Dr. Zsolt Kira
School of Interactive Computing
Georgia Institute of Technology

Dr. Rahul Sukthankar
Head of Perception
Google

Date Approved: March 4, 2020

Nobody ever figures out what life is all about, and it doesn't matter. Explore the world.

Nearly everything is really interesting if you go into it deeply enough.

Richard P. Feynman

To Aunt Robin who taught me the value of education

ACKNOWLEDGEMENTS

I have many people to thank for encouraging me during my studies at Georgia Tech. Firstly, I express my sincere gratitude to my advisor Dr. Irfan Essa for his support (and more importantly patience) of my research. I would also like to thank the rest of my thesis committee: Dr. Rahul Sukthankar, Dr. Judy Hoffman, Dr. Frank Dellaert, and Dr. Zsolt Kira.

I am lucky that my family has always been there to support me throughout my PhD as well. To my parents, Thomas and Laura, my grandmother Anita, and my brother Jonathan. My family was always there to give me advice on what to do and watch my dog when I went to the west coast for internships.

To my second family, the Silvas, Ella, Andrew, William, Lexi, Jeffrey, Tony, and Gabrielle who have helped me so much throughout my time at Georgia Tech.

To all my friends from Clemson my undergrad, who helped me continue my hobbies like climbing, and learn new things like distilling whiskey, especially Ryan, Sara, Brian, Taylor, Adam, and Beth Anne. As well as the many friends I made in Georgia and Tennessee, especially Josh, Charlie, Nate, Steve, Justin, both Jonathans and Allisons, Stephen, Sam, Meg, Kayla, Madison, Brandon, and many more.

My thanks also go out to all of my co-authors and mentors from Google Research/Brain. Dr. Vivek Kwatra, Dr. Avneesh Sud, Dr. Nicholas Dufour, Dr. Anelia Angelova, Dr. Rahul Sukthankar, Dr. Karthik Raveendran, Dr. Alireza Fathi, Dr. Kevin Murphy, and Dr. Matthias Grundmann. Matthias and Karthik were pivotal in mentoring me during the last years of my time as an intern in Google.

To my labmates and Georgia Tech friends, who I've lived with during internships and collaborated with over the years: Daniel Castro, Vinay Bettadapura, Edison Thomas, Yannick Schroecker, Pushkar Kolhe, Ashley Edwards, Himanshu Sahni, Sasha Lambert, Amir Shaban, Safoora Yousefi, Kalesha Bullard, Tesca Fitzgerald, Erik Wijmans, Meera Hahn,

Amit Raj, Cusuh Ham, Vince Cartillier, Apporva Beedu, Huda Alamri, Jonathan Balloch, Raphael Gontijo Lopes, Niranjan Kumar Kannabiran, and many more.

I'd also like to thank my previous advisors Dr. Henrik Christensen from Georgia Tech and Dr. Stan Birchfield from Clemson.

TABLE OF CONTENTS

Acknowledgments v

List of Tables xiii

List of Figures xvii

Chapter 1: Introduction 1

1.1 Thesis Statement 1

1.2 Thesis Contributions 2

1.3 List of Publications 5

Chapter 2: Background 7

2.1 Definitions 7

2.1.1 Segmentation 7

2.1.2 Object Retrieval 8

2.1.3 Depth and 3D 8

2.1.4 Surface Normals 9

2.1.5 Shape and Labels 10

2.1.6 3D Contextual Information 11

2.1.7 Encoding Information 12

2.2 Datasets 13
2.2.1 Scenenet RGBD 13
2.2.2 NYUDv2 14
2.2.3 Scannetv2 14
2.2.4 Cityscapes 15
2.3 Metrics 15

Chapter 3: Related Work 18

3.1 Unsupervised Segmentation 18
3.1.1 Image Segmentation 18
3.1.2 Video Segmentation 20
3.1.3 RGBD Video Segmentation 21
3.2 Unsupervised Object Clustering and Classification 22
3.2.1 Unsupervised Object Retrieval 22
3.2.2 Self Supervised Object Retrieval 23
3.3 Supervised Learning for Scene Understanding 25
3.3.1 Semantic Labeling 25
3.3.2 Surface Normal Estimation 26
3.3.3 Depth Estimation 28
3.4 Multi-Task Learning for Scene Understanding 29
3.4.1 3D Multi-Task Learning 31
3.4.2 Multi-Task Encoder Decoder Architectures 33

Chapter 4: Efficient Hierarchical Graph-Based Segmentation of RGBD Videos . 35

4.1 Introduction 35
4.2 Method 37
4.2.1 Spatiotemporal segmentation 38
4.2.2 Hierarchical Processing 40
4.2.3 Bipartite Graph Matching 41
4.3 Experimental Results 43
4.4 Conclusion 47

Chapter 5: Self-supervised Deep Clustering of Objects 49
5.1 Introduction 49
5.2 Approach 51
5.2.1 Self-supervised proposal generation 51
5.2.2 Unsupervised memory clustering (UMC) 53
5.3 Experimental evaluation 55
5.3.1 Unsupervised Memory-based clustering evaluation 56
5.4 CIFAR dataset experiments 57
5.5 Cityscapes experiments 58
5.5.1 Evaluation of the full unsupervised method 61
5.6 Conclusion 66

Chapter 6: Semantic Instance Labeling Leveraging Hierarchical Segmentation . 68
6.1 Introduction 68
6.2 Method 70
6.2.1 Fast Surface Normal Estimation 70

6.2.2 3D/4D Segmentation 71
6.2.3 Hierarchical Construction 71
6.2.4 Feature Selection 72
6.3 Experiments 74
6.3.1 4 Class Test 74
6.3.2 14 Class Test 75
6.3.3 Results 76
6.4 Conclusion 78

Chapter 7: Leveraging Semantics for Real-Time Surface Normal Predictions . . 80

7.1 Introduction 80
7.2 Method 82
7.2.1 Computing better ground truth normals 82
7.2.2 Datasets 82
7.2.3 Problems with current techniques 83
7.2.4 Improving Normals Using Point Clouds 85
7.2.5 Improving Normals with Semantics 85
7.2.6 Evaluation 86
7.3 Combining synthetic and real data 87
7.3.1 Mixing real and synthetic 87
7.3.2 Comparison with the state-of-the-art 88
7.4 Jointly predicting semantics and normals 90
7.4.1 Semantics 90

7.4.2 Joint Prediction 91
7.5 Training a real-time model 93
7.5.1 Model 94
7.5.2 Finetuning vs. Training from scratch 95
7.5.3 RGB vs. Grayscale 95
7.5.4 Network Size 95
7.5.5 Applications 97
7.6 Conclusion 98

Chapter 8: Sharing Decoders: Network Fission for Multi-task Pixel Prediction . 99

8.1 Introduction 100
8.2 Method 102
8.2.1 Backbone Architecture 102
8.2.2 Losses 104
8.2.3 Early Fission 105
8.2.4 Early-Mid and Mid Fission 106
8.2.5 Late Fission 106
8.2.6 Relationship to State-of-the-art Methods 106
8.3 Results 107
8.3.1 Surface Normal Ground Truth 108
8.3.2 Architecture Hyper-parameters 108
8.3.3 Fission-scheme Ablation Study 109
8.3.4 Variability Ablation Study 113

8.3.5 Auxiliary Loss Ablation Study 113
8.3.6 Loss Balancing Ablation Study 115
8.3.7 All 3 Modality Results 118
8.3.8 Real-world Results 120
8.4 Conclusion 122

Chapter 9: Conclusions 124
9.1 3D Contextual Information 125

Appendix A: Surface Normal Train/Test Evaluation Across Datasets 129

LIST OF TABLES

5.1 Contrastive loss on two embeddings on CIFAR10 for discovery of two classes. Per cluster accuracy (in %) for each of the two given classes on the test set (class labels are not used during the training process). 57

5.2 Unsupervised clustering results on CIFAR10 for discovery of two classes. Per cluster accuracy (in %) for each of the two given classes on the test set (class labels are not used during the training process). 58

5.3 Unsupervised clustering of objects from the Cityscapes dataset with 3 clusters with our clustering loss. The table shows number of examples assigned to each learned cluster (K=3). The clustering purity index is 61.99% for K=3 and the mean accuracy 76.16%. The baseline index is 33.27% and 50.61%, respectively. 61

5.4 Unsupervised clustering results on Cityscapes dataset. Clustering purity index for K=3 on the validation set (class labels are used in evaluation only). 61

5.5 Recall of our proposal generation algorithm on Cityscapes objects (in %). Bottom row shows all examples total available in the data. 63

5.6 Our clustering on Cityscapes validation set with unsupervised proposals. . . 63

5.7 Baseline clustering on Cityscapes validation, unsupervised proposals without our loss. 64

5.8 Clustering on Cityscapes validation, unsupervised proposals with supervised features 65

5.9 Clustering results with unsupervised proposals. Clustering Purity Index. Our unsupervised method outperforms the baseline method which uses the same inputs. We compare to clustering with supervised features as an upper bound. 65

6.1 A comparison of the per-pixel and per-class classification accuracy of the 4 class set comparing our algorithm to other state of the art methods. 77

6.2 A comparison of the per-pixel and per-class classification accuracy of the 14 class set proposed by [78] comparing our algorithm to other state of the art methods. 78

7.1 Accuracy and error rates of our normals model (without semantic output head) when evaluated on the NYUDv2 normals from [86]. First row shows results using our simple network training on standard normals as shown in the top right of Figure 7.1. Second row shows results using our denoising method (bottom left of Figure 7.1). Third row shows results using our semantic smoothing method (bottom right of Figure 7.1). Evaluation is performed on the normals from [86], which demonstrates the necessity of a dataset cleanup. 86

7.2 Comparison against state of the art on surface normal estimation task. All methods (except in the last row) are evaluated on the normals from the NYUDv2 dataset computed using the method of [86]; in the last row, we show our method evaluated on the normals from NYU computed using our method, which we denote by NYU'. 88

7.3 Normal Accuracy comparisons with different testing and training datasets. The columns are the training set used and the rows are the evaluation accuracy for each individual dataset. Scenenet+NYU FT means the standard practice of pretrained on synthetic and finetuned on NYUDv2. Datasets Mixed means the dataset is trained all from scratch with a batch-wise mix. The best result for each row is bold. 89

7.4 This shows the effect of multiplying the cosine loss learning rate. 92

7.5 This table shows the different effect of the learning rates of the semantic trained network. 93

7.6 This shows the results of Joint semantics and normals prediction on Scannet. The Normals column is the accuracy of a network only trained on Scannet normals. The semantics is the accuracy when only trained on Scannet semantics. Joint is the accuracy when both are trained concurrently as per Section 7.4. 94

7.7 This ablation study shows the effect of color on the network by changing the percent of input training images that are converted to grayscale. Interestingly enough, color does not seem that important for surface normal estimation. 96

7.8 Here we use an ablation study to test performance vs network size. The channel multiplier is a multiplier that determines the number of output channels calculated at each block. For instance, the final output of the encoder at channel multiplier 32 has 1280 channels, whereas, at channel multiplier 16, it has 640 channels. 97

8.1 Here are the individual results for each modality on the Scenenet dataset. For this, each method is trained and evaluated on only one task which is why we indicate - on the others. On each of these results, higher is better. . 108

8.2 Joint Normals and Semantics results with different fission methods when the loss is balanced ($\lambda_{\text{normals}} = 10.0$ and $\lambda_{\text{semantics}} = 1.0$) and auxiliary loses are used ($\alpha_1 = 0.6, \alpha_2 = 0.5$ for both tasks) with different initialization methods. Imp is the percent improvement metric and PIPFIP is percent improvement per flop increase percentage. 109

8.3 Joint Normals and Depth results using mid fission with loss balancing ($\lambda_{\text{normals}} = 10$ and $\lambda_{\text{depth}} = 1$). Here we compare when trained from 3 different initializations: Scratch being ImageNet fine-tuning, Normals being initialized with the network trained on normals, Depth being initialized with the network trained on depth. For this ablation, auxiliary loses are used ($\alpha_1 = 0.6, \alpha_2 = 0.5$ for both tasks). 111

8.4 Joint Semantics and Depth results using mid fission with loss balancing ($\lambda_{semantics} = 2$ and $\lambda_{depth} = 1$). Here we compare when trained from 3 different initializations: Scratch being ImageNet fine-tuning, Normals being initialized with the network trained on normals, Depth being initialized with the network trained on depth. For this ablation, auxiliary loses are used ($\alpha_1 = 0.6, \alpha_2 = 0.5$ for both tasks). 112

8.5 3 Different training runs of our mid-fission network on normals and semantics with normal initialization and balanced losses. 113

8.6 Joint Normals and Semantics results with different aux losses using our proposed mid fission without loss balancing. 114

8.7 Joint Normals and Semantics results with different loss balancing using our proposed mid fission. 115

8.8 Joint Normals and Depth results with different fission methods both with no loss balancing and when the loss is balanced ($\lambda_{normals} = 10$ and $\lambda_{depth} = 1$). For this ablation, auxiliary loses are used ($\alpha_1 = 0.6, \alpha_2 = 0.5$ for both tasks). 116

8.9 Joint Depth and semantic label results with different fission methods with no loss balancing and when the loss is balanced ($\lambda_{semantics} = 2$ and $\lambda_{depth} = 1$). For this ablation, auxiliary loses are used ($\alpha_1 = 0.6, \alpha_2 = 0.5$ for both tasks). 117

8.10 Joint 3 task results with different fission methods both with no loss balancing and when the loss is balanced ($\lambda_{semantics} = 1$, $\lambda_{depth} = 0.5$ and $\lambda_{normals} = 10$). E-Mid is Early-mid, N is the single task normals baseline, D the depth, and S the semantics. 119

8.11 NYUDv2 13 class results with our proposed mid fission method compared to state of the art training on NYUDv2 13 class. 120

8.12 NYUDv2 40 class results with our proposed mid fission method compared to state of the art (fine-tuned). - means the method does not train and evaluate on that task. * Indicates the normals are from [89] that were then quantized to RGB images. 121

8.13 Scannetv2 validation set results. with our proposed mid fission method compared to state of the art. - means the method does not train and evaluate on that task. 122

A.1 Normal Accuracy comparisons with different testing and training datasets. The columns are the training set used and the rows are the evaluation accuracy for each individual dataset. *Scenenet NYU FT* means pretrained on Scenenet and finetuned on NYUDv2. *Scenenet+NYU* means the the dataset is trained with a batch-wise mix. *Scenenet+Scannet+NYU* means a batch-wise mix with all 3 datasets. *Average eval* is the evaluation for NYU, Scannet, and Scenenet averaged for each training split. The best result for each row is bold. 130

LIST OF FIGURES

2.1 An example of segmentation. On the left is an input RGB image and on the right is a random colorized segmentation output. 7

2.2 An example of object retrieval where the left is an input image and the right is a set of masked objects retrieved from a background and colorized. 8

2.3 An example of a point cloud from different viewpoints. 9

2.4 An example of surface normals with an RGB image on the left and colorized surface normals on the right. Lime green is up, pink is right, dark green is left, light purple is forwards, and dark purple is downwards. 10

2.5 An example of what we mean when we use the terms shape and labels. . . . 11

2.6 Our definition of 3D contextual information. 12

2.7 How do we encode 3D contextual information? 13

2.8 The Scenenet RGBD dataset from [14]. 13

2.9 The NYU Depth V2 RGBD dataset from [15]. 14

2.10 The Scannetv2 RGBD dataset from [16]. 14

2.11 The Cityscapes outdoor dataset from [17]. 15

3.1 An example of segmentation from Felzenszwalb [18]. 19

3.2 An example of segmentation from Grundmann [19]. 20

3.3 An example of segmentation from Papon [24]. 21

3.4 An example of a method for unsupervised discriminative patches using clustering from Singh [52]. 23

3.5 An example of a method for self-supervised clustering from video by Pathak [45]. 24

3.6 An example of a method for supervised semantic labeling from Gupta [72]. 25

3.7 An example of a method for deep surface normal estimation from RGB images from Bansal [83]. 27

3.8 The loss module from Godard [94] using left and right disparity maps d^l and d^r. Here I^l and I^r are the left and right images, C=Convolution, UC=Up-Convolution, S=Bilinear sampling, US=Upsampling, and SC=Skip connection. 28

3.9 Eigen's method from [89] for a single network that can be adapted to do 3 different tasks (though not concurrently) showing that neural networks can learn different tasks individually without changing the architecture. 30

3.10 Misra's method from [101] for a soft-parameter sharing method that stitches two individual tasks together. 31

3.11 An example of a taskonomy from Zamir[108] for task-relatedness. 32

3.12 An example from Kendall [114] showing a multi-task encoder-decoder architecture using what we call *early fission* with a special multi-task loss. . . 33

4.1 Segmentation of streaming 3D Point clouds. Top: RGB and depth maps used as input in our algorithm. Bottom: Rendered version of the point cloud and its frame segmentation output. 36

4.2 Schematic of our method to segment streaming 3D point clouds. 37

4.3 Top row: original RGB image (depth image not shown), and ground truth object boundaries. Middle three rows: The output of the graph-based algorithm using a linear combination of color and depth, with α values of 0.1, 0.2, 0.4, 0.6, 0.8, and 0.9; none produces a correct result. Bottom row: The output of the initial over-segmentation step of our algorithm (left), as well as the boundaries of this output (right). 44

4.4 Left: The error based on the choice of α on the output of the linear combination algorithm on three sequences (blue lines), as well as the output of our hierarchical algorithm (red lines), which is not dependent on α Right: Effect of varying k_{color} on the output of our initial over-segmentation step, as well as the value of k on the output of [18] (with the best α possible. . . . 45

4.5 Left: Effect of varying ζ on the output of our hierarchical clustering applied to the output of either our initial over-segmentation step or that of [18] (with the best α and k possible). Right: The explained variation from [124] of our method, [124], and [19] on Sequence 2. 45

4.6 Sequence 1, (A) : RGB image, (B): Results from [124], (C): Results from [19] (N/A is used when the frame is beyond the length it can run), (D): Our Results 46

4.7 Sequence 2, (A) : RGB image, (B): Results from [124], (C): Results from [19] (N/A is used when the frame is beyond the length it can run), (D): Our Results 47

4.8 Sequence 3, (A) : RGB image, (B): Results from [124], (C): Results from [19] (N/A is used when the frame is beyond the length it can run), (D): Our Results 48

5.1 An overview of the method: self-supervision is used to learn proposals for full objects; unsupervised deep clustering is then used to uncover semantic object classes. 50

5.2 Schematic architecture which generates object proposals with self-supervised methods (e.g. learning of depth from video). The depth prediction convolutional stack is based on VGG [69] 51

5.3 Unsupervised Memory Clustering with the clustering objective. Clusters will be learned as additional weights, jointly with the feature embedding for positive samples. 52

5.4 Number of examples per class. Cityscapes dataset. 56

5.5 The clustering of the dog class (Class 1 in CIFAR) and automobile class (Class 5 in CIFAR) while training on the CIFAR10 dataset. The full training process is shown until about 12,000 steps. As seen, the newly formed cluster 0 (first two figures to the left), tends to attract much of the dogs examples, Class 1 in CIFAR, (first figure), than automobile, or Class 5, examples (second figure). At the same time, the newly learned cluster 1, does not classify correctly dog examples (third figure), but does well on automobiles (fourth figure). Blue/Orange curves denote training vs. test data. The training has no notion of labels, they are used here only for evaluation purposes. 58

5.6 Visualization of clusters learned by UMC when proposals are given (for K=3). It is clear that the green class is responsible for retrieving cars, the blue one persons, and the red one bicycles. We can see that both cars and persons are discovered pretty well. Bicycle is much more rare class and can be confused with a person or with a partially visible car in the background. (Best viewed in color.) 59

5.7 Example proposal instances generated from unsupervised depth and clustering in RGBD space. 62

5.8 Main algorithm results: Visualization of clusters learned by the unsupervised procedure when the proposal boxes are also unsupervised. Several interesting clusters are formed: person is retrieved and clustered in the red cluster; side and back view of a car tends to be in the blue cluster, and the green cluster contains buildings,other structures, and also the front view of cars. Some noisy classifications and proposals are also present (e.g. right row, bottom two). Best viewed in color. 67

6.1 Semantic segmentation and labeling of 3D Point clouds. Top: RGB and depth maps used as input in the algorithm. Bottom: The 4 Class output and 10 Class output as described in Section6.3 69

6.2 An overall schematic of our full method. First we generate the point cloud and a graph, then we generate surface normals, then we over segment the point cloud, create a hierarchy of super pixels, and finally train our feature vectors using a random forest of decision trees. 70

6.3 Classification Accuracy for different 3D Features on a validation set as described in Section 6.3. Heat mapped with blue being 0% and red being 100% 73

6.4 Confusion Matrix for 4 classes of NYU Dataset Left: [15], Center: our method with SIFT, Right: our method without SIFT. Heat mapped with blue being 0% and red being 100% 75

6.5 Confusion Matrix for the top 13 classes of NYU Dataset[15] as proposed by [78] (ignoring the 14th class other). Heat mapped with blue being 0% and red being 100% 76

6.6 Results for the 14 class set proposed by [78] The top row shows RGB, the middle row shows the ground truth labels, and the bottom row shows our classification. 77

6.7 Results from the 4 Class set from [15]. The top row shows RGB, the middle row shows the ground truth labels, and the bottom row shows our classification. 79

7.1 Visualization of different ways of computing "ground truth" normals. Top left: a sample image from the NYUDv2 dataset. Top-right: computed using method of [89] that uses a small smoothing window. Bottom-left: results of our method using larger smoothing window. Bottom-right: results of our method after semantic smoothing (if labels are available). Note that the back and right wall are cleaned up to a large degree due to this correction. . 81

7.2 Illustration of the NYU data, together with the predictions of our model on them. Columns, from left to right: RGB image, depth, ground truth surface normals, our predictions, error image (where black is under 11.25 degrees errors, and then error increases from yellow to purple). 83

7.3 Qualitative evaluation on the Scannet data. Columns are the same as in Figure 7.2. 83

7.4 A common example of the errors seen in the normals from [86] that many use for training and evaluation. In these visualizations, (r, g, b) map to (x, y, z) of the normal at that location. Note the over-smoothing, which reduces and removes the normals of small objects. This image also demonstrates why it is important to only backpropagate on pixels that have valid depth, as the right side of the image has incorrect normal data due to noisy and missing depth values. 84

7.5 Effect of window smoothing size on surface normals. a) RGB image. b) Normals computed with a normal smoothing size of 10. c) In-painted depth. d) Normals computed similar to [15] from (c) using window size 10. e) The mask generated for valid training pixels. f) The normals with window size 30 that we train on (backpropagating only when e is valid). 84

7.6 Qualitative evaluation on Scenenet data. Columns, from left to right: RGB image, depth, ground truth surface normals, our predictions, error image (where black is under 11.25 degrees errors, and then error increases from yellow to purple). 87

7.7 Examples of our semantic labeling predictions for the Scannet dataset. From left to right: RGB image, ground truth, our predictions. 90

7.8 Examples of our semantic labeling predictions for the Scenenet dataset. From left to right: RGB image, ground truth, our predictions. 91

7.9 Our Architecture for joint prediction involves a shared encoder inspired by Mobilenet [142], followed by two U-net decoders. Each outputs its prediction and has a separate loss for either segmentation or normal prediction. The losses along with regularization are summed and optimized jointly. When doing just normal prediction, we simply drop the segmentation decoder and loss. See Section 7.5 for details. 92

7.10 A sample AR application that uses the surface orientation to place a virtual character and text. 98

8.1 The possible fissions for the eASPP decoder part of the architecture[157]. The blue block is the encoder bottleneck, the orange blocks are the first two decoder blocks, which include convolutional, batchnorm, and relu layers with 256 output channels and skip connections. The purple blocks are the last decoder block which has convolutional transpose, batchnorm, and relu layers. 103

8.2 Illustration of the Scannetv2 data, together with the predictions of our model on them. Columns, from left to right: RGB image, ground truth surface normals, our normal predictions, ground truth semantic labels, then our semantic predictions. 123

SUMMARY

This thesis aims to demonstrate how using 3D cues improves semantic labeling and object classification. Specifically, we will consider depth, surface normals, object classification, and pixel-wise semantic labeling in this work. The works outlined in this document aim to validate the following thesis statement: **Shape, used as an additional context, improves segmentation, unsupervised clustering, object classification and semantic labeling with little computational overhead.**

The thesis will show that: **Combining shape and object labels improves results while (1) requiring few extra parameters, (2) provides better results using surface normals than depth, and (3) combining shape with labels improves accuracy for each task.** We describe various methods to combine shape and object classification and then discuss our extensions of the work which focus on surface normal prediction, depth prediction, and semantic labeling specifically.

In Chapter 4, we present an efficient algorithm for segmenting 3D pointclouds over time by combining color and depth cues in a manner that respects depth boundaries to improve unsupervised segmentation. Chapter 6 presents an approach extending the method from Chapter 4 to classify each segment as label in an indoor environment using both shape and color features. Chapter 5 presents an approach using the color and depth segmentation from Chapter 4 as an input for deep unsupervised clustering. Chapter 7 introduces a method using semantic labels to both improve ground truth surface normals and to improve prediction of surface normals with a real-time encoder-decoder neural network. Finally, Chapter 8 introduces a shared encoder-decoder architecture and the concepts of network fission as well as ablation studies on which hard-parameter encoder-decoder fission is best.

The earlier chapters introduce, motivate, and provide background related work for this dissertation.

CHAPTER 1
INTRODUCTION

Humans have an inherent understanding of the shape of their environment and the objects contained in it. Given a description of a room, a person can understand a reasonable approximation of the space and the objects. However, our current methods lack this type of contextual understanding (i.e. a chair is shaped a particular way and indicates you can sit on it). This work is motivated by the idea that there is an inherent relationship between 3D information such as shape and scene understanding/object classification. Objects such as tables, chairs, and cups have a specific shape and our models should leverage and learn that information. Depth and surface normals have frequently been used as additional signals in semantic labeling work; however, there is still limited understanding on the best ways of using and learning shape and labels jointly. Our work examines using 3D cues for unsupervised and supervised approaches for segmentation and semantic labeling. We show how to use 3D information for robust unsupervised segmentation, supervised semantic labeling using segmentation, and unsupervised object categorization. We explore this relationship further by showing how shape helps deep neural networks semantically label indoor environments. We explore how joint estimation of shape and labels improves both results when learned together and how to accomplish both with little added model capacity. More specifically, we aim to validate the following theses and hypothesis:

1.1 Thesis Statement

Shape, used as an additional context, improves segmentation, unsupervised clustering, object classification and semantic labeling with little computational overhead. Combining shape and object labels improves results while (1) requiring few extra pa-

rameters, (2) provides better results using surface normals than depth, and (3) combining shape with labels improves accuracy for each task.

1.2 Thesis Contributions

.

1) Combining color and depth cues for unsupervised segmentation:
We present an efficient and scalable algorithm for segmenting 3D RGBD point clouds by combining depth, color, and temporal information using a multistage, hierarchical graph-based approach. Our algorithm processes a moving window over several point clouds to group similar regions over a graph, resulting in an initial over-segmentation. These regions are then merged to yield a dendrogram using agglomerative clustering via a minimum spanning tree algorithm. Bipartite graph matching at a given level of the hierarchical tree yields the final segmentation of the point clouds by maintaining region identities over arbitrarily long periods of time. We show that a multistage segmentation with depth then color yields better results than a linear combination of depth and color. Due to its incremental processing, our algorithm can process videos of any length and in a streaming pipeline. The algorithm's ability to produce robust, efficient segmentation is demonstrated with numerous experimental results on challenging sequences from our own as well as public RGBD data sets.

[1]: S. Hickson, S. Birchfield, I. Essa, and H. Christensen, "Efficient hierarchical graph-based segmentation of rgbd videos," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 344–351

2) Leveraging color and depth segmentation for deep unsupervised clustering:
We consider the problem of retrieving objects from video sequences and learning to classify them into semantic categories without manual supervision. This work is motivated by the observation that video contains significant object motion and inherently a dynamic

3D scene structure that can be exploited for self-supervision learning and classification. Our approach is based on two key ideas: (1) We propose to estimate the depth of the scene and cluster it in RGBD space, to retrieve potential object candidates; here depth is learned in unsupervised manner from video sequences. (2) We build simultaneous unsupervised clustering and learning of the feature embedding of the object proposals.The latter is accomplished by a fully differentiable unsupervised deep clustering objective, where the clusters are learned as parameters of the network and are represented as memory units. Our results demonstrate that this technique creates useful embeddings for subsequent clustering. Our experiments on the challenging Cityscapes dataset show meaningful retrieval of objects' proposals without supervision, and clustering into classes that visually correspond to common categories in scenes.

[2]: S. Hickson, A. Angelova, I. Essa, and R. Sukthankar, "Unsupervised deep clustering for semantic object retrieval," in *Bay Area Machine Learning Symposium (BayLearn)*, http://www.baylearn.org/, 2017

[3]: S. Hickson, A. Angelova, I. Essa, and R. Sukthankar, "Object category learning and retrieval with weak supervision," in *NeurIPS Learning with Limited Labeled Data Workshop*, IEEE, 2018, pp. 1068–1075

[4]: S. Hickson, A. Angelova, I. A. Essa, and R. Sukthankar, *Category learning neural networks*, US Patent App. 16/511,637, 2020

3) Leveraging color and depth segmentation for semantic labeling:

Most of the approaches for indoor RGBD semantic labeling focus on using pixels or superpixels to train a classifier. In this chapter, we implement a higher level segmentation using a hierarchy of superpixels to obtain a better segmentation for training our classifier. By focusing on meaningful segments that conform more directly to objects, regardless of

size, we train a random forest of decision trees as a classifier using simple features such as the 3D size, LAB color histogram, width, height, and shape as specified by a histogram of surface normals. We test our method on the NYU V2 depth dataset, a challenging dataset of cluttered indoor environments. Our experiments using the NYU V2 depth dataset show that our method achieves state of the art results on both a general semantic labeling introduced by the dataset (floor, structure, furniture, and objects) and a more object specific semantic labeling. We show that training a classifier on a segmentation from a hierarchy of super pixels yields better results than training directly on super pixels, patches, or pixels as in previous work.

[5]: S. Hickson, I. Essa, and H. Christensen, "Semantic instance labeling leveraging hierarchical segmentation," in *2015 IEEE Winter Conference on Applications of Computer Vision*, IEEE, 2015, pp. 1068–1075

4) Leveraging semantics for real-time surface normal prediction:

We propose 4 insights that help to significantly improve the performance of deep learning models that predict surface normals and semantic labels from a single RGB image.These insights are: (1) denoise the "ground truth" surface normals in the training set to ensure consistency with the semantic labels; (2) concurrently train on a mix of real and synthetic data, instead of pre-training on synthetic and fine-tuning on real; (3) jointly predict normals and semantics using a shared model, but only back propagate errors on pixels that have valid training labels; (4) slim down the model and use grayscale instead of color inputs. We demonstrate consistently improved state of the art results on several datasets, using a model that runs at 12 fps on a standard mobile phone.

[6]: S. Hickson, K. Raveendran, A. Fathi, K. Murphy, and I. Essa, "Floors are flat: Leveraging semantics for real-time surface normal prediction," in *The IEEE International Conference on Computer Vision (ICCV) Workshops Geometry Meets Deep Learning (GMDL)*, 2019

(5) Learning deep shared features for semantic labeling, surface normal prediction, and depth prediction:

We show that surface normals and other shape based encodings are better than raw depth for improving semantic labeling and classification when trained jointly. This is different than using surface normals or a shape representation as an intermediate step or input, which has been explored at length. We experiment exploring different possible decoder splits for network fission in the decoder to determine where the best method lies for shape plus semantic labels. We further propose a model that jointly learns depth, surface normals, and semantic labels using different decoder fission splits. We experiment with sharing the encoder network architecture for joint prediction, but also most of the decoder (we call this mid-fission). We show that object labeling and shape are closely related. Due to this, a shared encoder-decoder with two small separate one-layer heads produces better results than the conventional method of sharing an encoder and training separate full decoders.

1.3 List of Publications

- An energy minimization approach to 3D non-rigid deformable surface estimation using RGBD data[7]
- **Efficient hierarchical graph-based segmentation of RGBD videos[1]**
- **Semantic instance labeling leveraging hierarchical segmentation[5]**
- Predicting daily activities from egocentric images using deep learning[8]
- **Unsupervised deep clustering for semantic object retrieval[2]**
- **Object category learning and retrieval with weak supervision[3]**
- **Category learning neural networks[4]**

- Let's Dance: Learning From Online Dance Videos[9]
- Expressions in VR Using Eye Tracking Cameras[10]
- Eyemotion: Classifying facial expressions in VR using eye-tracking cameras[11]
- Classifying facial expressions using eye-tracking cameras[12]
- **Floors are Flat: Leveraging Semantics for Real-Time Surface Normal Prediction[6]**

This thesis will discuss the papers above highlighted in bold as well as the final work. Chapter 4 will cover [1]. Chapter 5 will cover [2], [3], and [4]. Chapter 6 will cover [5]. Chapter 7 will cover [6], Finally, Chapter 8 will cover the final work and Chapter 9 will conclude the discoveries of the previous chapters.

CHAPTER 2

BACKGROUND

The focus of this dissertation is to use 3D contextual information to improve segmentation, object retrieval, shape prediction, and semantic labeling. As such, the approaches in later chapters will build off of some common themes including specific datasets, metrics, and ideas. We cove some of the related background work and terms in brief in this chapter.

2.1 Definitions

First, we go through some common definitions of computer vision tasks we use in this dissertation.

2.1.1 Segmentation

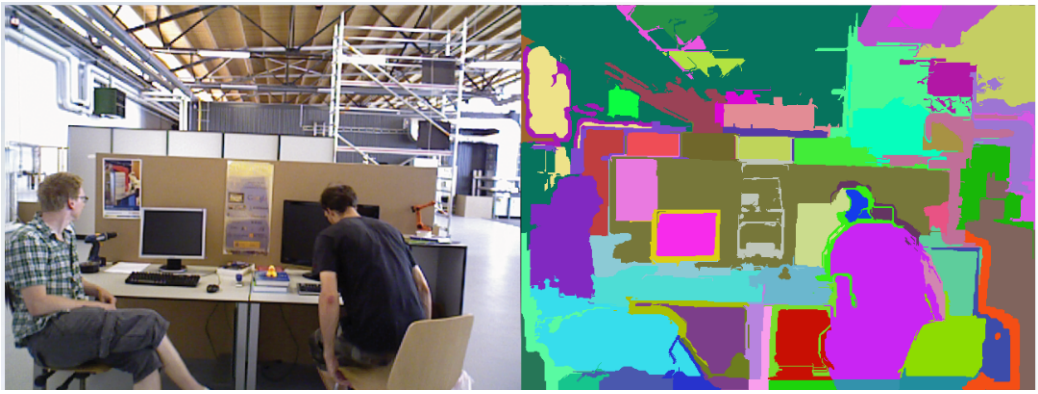


Figure 2.1: An example of segmentation. On the left is an input RGB image and on the right is a random colorized segmentation output.

Image segmentation is an important computer vision problem that spans back over 40 years[13]. The goal is to group perceptually important elements together into a cohesive and meaningful region. An example is shown in Figure 2.1 where the segmentation output

labels are colorized for easy viewing. Note that perceptually similar regions group together as one color. The algorithm doesn't necessarily know what each region is, just that it's independent and visually distinct from other regions. The knowledge of what the region is (i.e. chair, floor) makes it *semantic segmentation*.

2.1.2 Object Retrieval

Object retrieval is harder to define than segmentation. It involves extracting "important" regions from an image. These regions can be represented as a bounding box or a segmentation. An example of a segmentation object retrieval is shown in Figure 2.2 where the method is extracting close people, cars, and bicycles in the foreground from the background. Many of these methods tend to be unsupervised or self-supervised. Some object retrieval methods don't aim for high accuracy at all but just recall with the idea being as long as all the important regions are proposed, some other mechanism can filter them. Similar to segmentation, typically there is no algorithmic knowledge of what the object is, if there is, we call it semantic object retrieval or object detection.

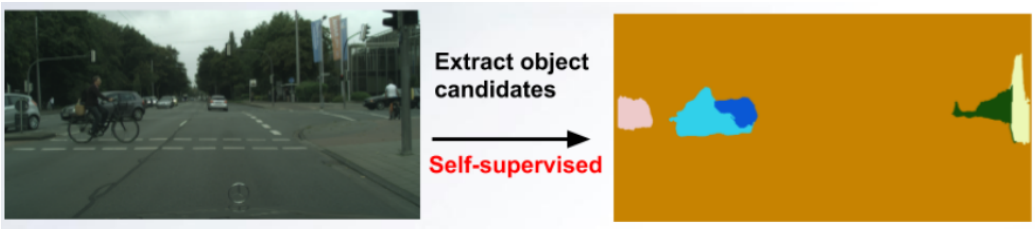


Figure 2.2: An example of object retrieval where the left is an input image and the right is a set of masked objects retrieved from a background and colorized.

2.1.3 Depth and 3D

All of the work we discuss in this dissertation will involve 3D environments, meaning that instead of a 2D image with x, y pixels, we have a 3D space with x, y, z voxels (volumentric pixels). In some of this work, we will extend this to x, y, z, t for toxels (temporal voxels).

Much of the data we use is collected from projective and time of flight depth sensor devices such as the Microsoft Kinect v1/2. These devices give a 2D depth map similar to an RGB image where each pixel is instead the depth from the camera. In these cases, our z coordinate comes from a depth frame and we have a 2.5D representation, which can be seen as a slice of an environment from a certain viewpoint. An example of this is shown in Figure 2.3 where 3 viewpoints of a 2.5D environment are shown. It can be seen that occluded regions from the original viewpoint are never seen; we simply know how far away each *observed* pixel is. We can use the RGB Depth pairs with known camera calibration information to construct point clouds (a 3D representation of our image) as shown in Figure 2.3.

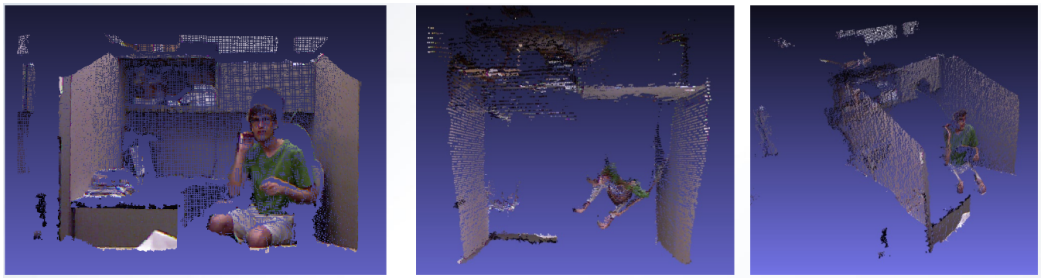


Figure 2.3: An example of a point cloud from different viewpoints.

2.1.4 Surface Normals

The surface normal of a region is a vector that is perpendicular to the region. For example, the normal of a ceiling is pointing directly down, a floor directly up, a wall to your left directly right, and a wall to your right directly left. Surface normals are frequently used for AR/VR applications, graphics, gaming, and visualizations. For instance, they tell us what direction a ball will bounce when it collides with an object. In our work, we construct a surface normal for each pixel/voxel. For much of our work, we consider surface normals in the camera's point of view; meaning, which way is this surface pointing from what the camera can see. If the camera is pointing at a floor, the surface normal will be directly towards the camera. If it is pointing towards a floor or ceiling, it will be the same.

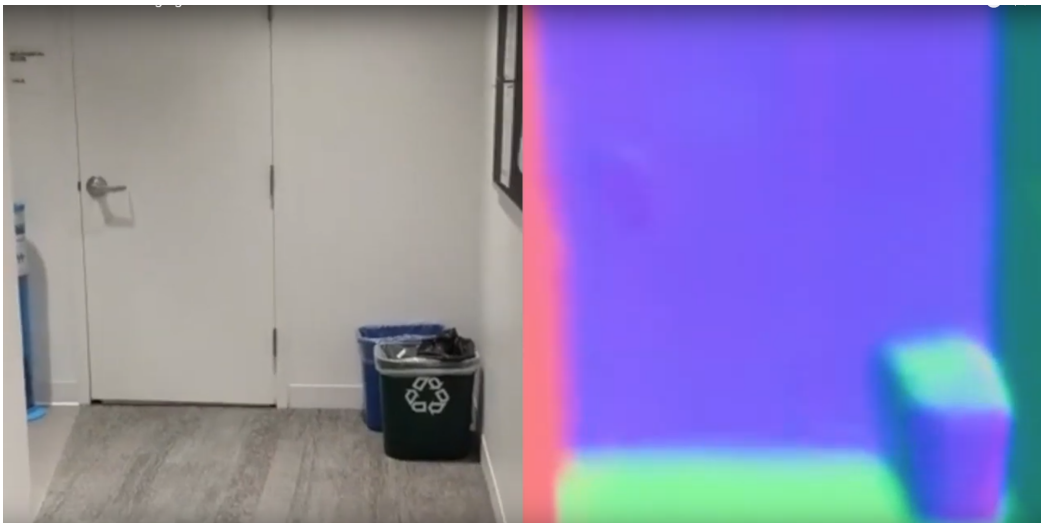


Figure 2.4: An example of surface normals with an RGB image on the left and colorized surface normals on the right. Lime green is up, pink is right, dark green is left, light purple is forwards, and dark purple is downwards.

Unlike other ground truth, surface normals must be calculated and are inherently noisy due to the noise from depth and the types of calculations that are used to find them. A trivial solution is just to select many neighbors and find the least-squares solution; however, methods frequently must do more than that to account for depth noise and the distance each object is from the camera. In our work, all surface normals are unit vectors. A visualization of surface normals is shown in Figure 2.4.

2.1.5 Shape and Labels

We frequently use the terms shape and labels in this dissertation. Here, shape means some representation of the 3D shape of an object or scene. This could be surface normals, depth, 3D position, curvature, etc. An example of this is shown in Figure 2.5 where the couches are represented with their surface normals. In this case, the normals indicate that the couches are on a planar surface and have a specific shape, which includes armrests and a back. When we say labels, we mean some time of classification or semantic labeling of an object or scene. For instance, in Figure 2.5, we are told that those pixels are indeed

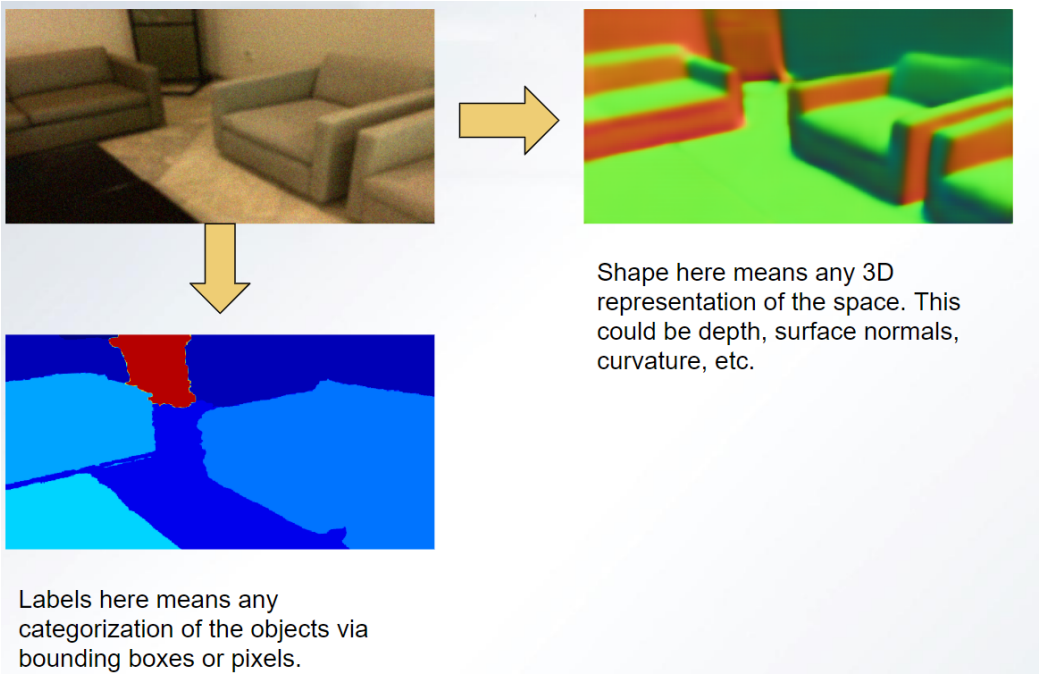


Figure 2.5: An example of what we mean when we use the terms shape and labels.

couches and they sit on top of a floor around a table in the corner of a room.

2.1.6 3D Contextual Information

3D contextual information is another term we use frequently in this dissertation that can be hard to fully define. Context can be defined here as *circumstances that help form understanding for an environment, statement, or idea*. 3D contextual information is information about a 3D scene that can be leveraged to further understanding. An example of this is shown in Figure 2.6. We are given shape and labels of some couches. We know from context that couches are on floors and often around tables; we also know that couches can be sat in and are therefore normally pointing up. Humans have learned this inherent understanding of an environment and can leverage it appropriately. Any object pointing up that is comfortable, a reasonable height, and structurally sound can be used as a chair. That is something we as humans all know that our algorithms currently cannot learn. Contextual information is how we bridge that gap.

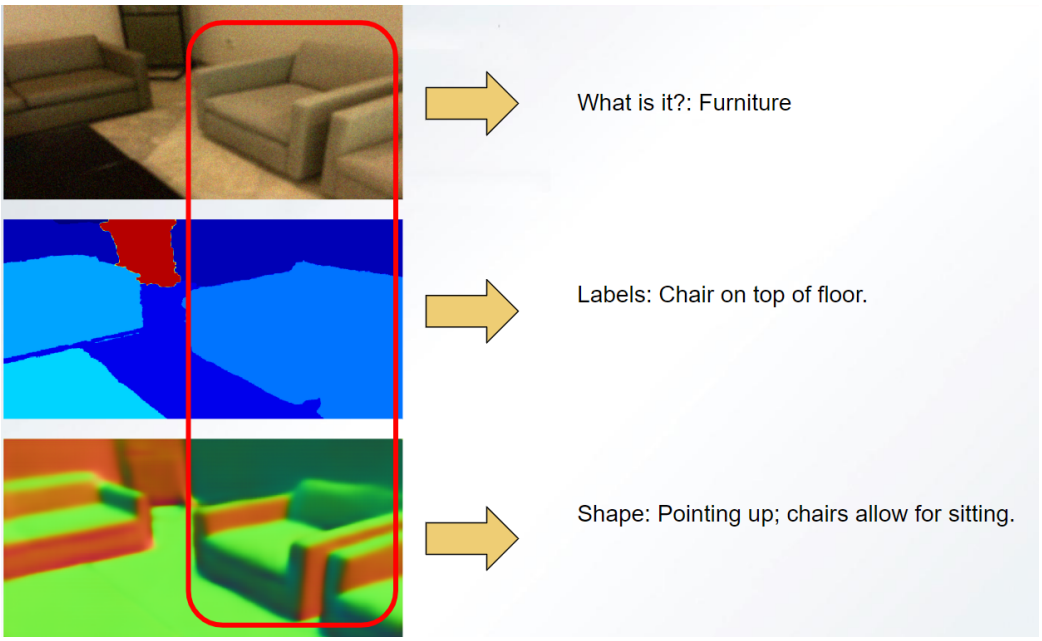


Figure 2.6: Our definition of 3D contextual information.

2.1.7 Encoding Information

The question then becomes, how do we encode that information? If we just have some vector of numbers, how do we learn this context, and how can we use it to understand the environment? In Figure 2.7, we show a scene with some information given to an algorithm (bounding boxes labeled recycle bins and door). How do we encode knowledge in some way for a computer to understand the door is in fact a door even though it's visually the same color as the wall; or that compost/recycle bins sit on the floor? That is the focus of this thesis.

This work is motivated by the idea that there is an inherent relationship between 3D information such as shape and scene understanding/object classification when viewing dynamic environments.

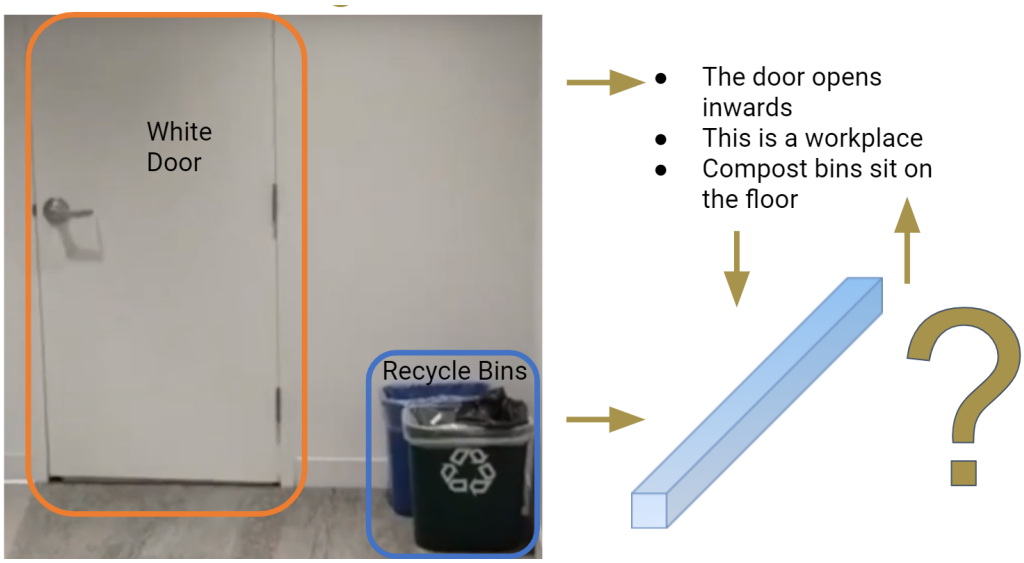


Figure 2.7: How do we encode 3D contextual information?

2.2 Datasets

Below we discuss the datasets used throughout this dissertation. Our work has focused on a small number of datasets we will discuss briefly below.

2.2.1 Scenenet RGBD

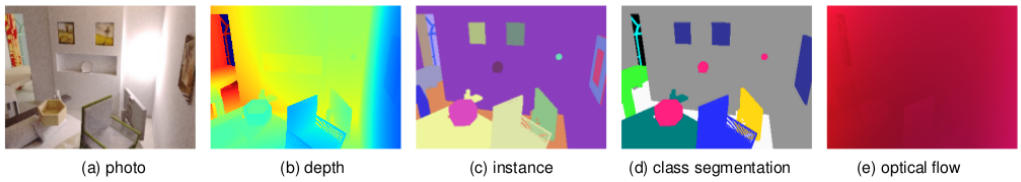


Figure 2.8: The Scenenet RGBD dataset from [14].

Scenenet RGBD is a set of 5 million semi-photorealistic synthetic RGBD images from more than 15,0000 trajectories of synthetic layouts with random object poses with random lighting and texture synthesis. Extracted from this are ~ 4 million 320x240 images with corresponding camera parameters, depth, and semantics; we can also extract surface normals from the depth.

2.2.2 NYUDv2

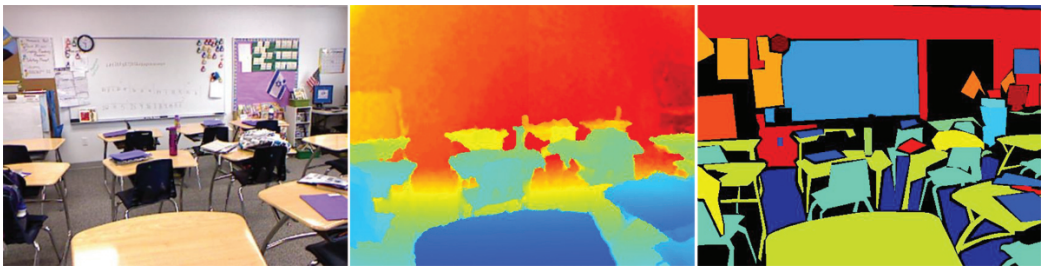


Figure 2.9: The NYU Depth V2 RGBD dataset from [15].

NYUDv2 [15] is a dataset of indoor environments taken with a Kinect device which results in approximately 450,000 640x480 RGB-depth pairs. Semantic labels are created for 1449 of these images, split into predefined train and test sets. There are 13 and 40 class label splits that literature uses for comparisons.

2.2.3 Scannetv2

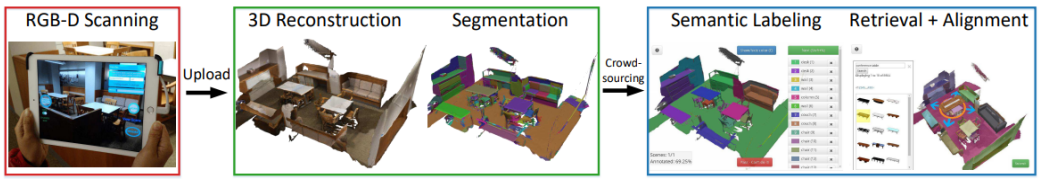


Figure 2.10: The Scannetv2 RGBD dataset from [16].

Scannetv2 [16] is a dataset of approximately 2 million 1296x968 RGB images with 640x480 depth sensor images with included pixel-level semantic segmentation labels. Instead of 2D annotation like in NYUDv2, these labels were annotated in 3D and then back-projected into 2D, which allows for more labeled images but the annotations can be less accurate and edges sometimes don't match up perfectly. Given the frames are taken from video, there is also a lot of repeated information in the dataset, so random sub-sampling is an effective strategy to construct a good training set.

2.2.4 Cityscapes

Figure 2.11: The Cityscapes outdoor dataset from [17].

Cityscapes dataset: The Cityscapes dataset [17] is a challenging, large-scale dataset that is used for evaluating various classification, detection, and segmentation algorithms related to autonomous driving. It is collected in 50 different cities with a core set of 5000 different scenes. The data is split in standard train, validation, and test sets according to a predetermined protocol. It contains 2975 training, 500 validation, and 1525 test high resolution 2048x1024-pixel images, where the test set is provided for the purposes of the Cityscapes competition only.

2.3 Metrics

For depth metrics, we use the well established relative depth metric in Equation 2.1 where δ is 1.25, 1.25^2, and 1.25^3. These numbers are based on human perception of depth differences. y is the ground truth pixel-wise depth and z is the final output of the task decoder. Some methods also include relative depth and root-mean squared error which are also easy to compute but for ease of reading we have avoided these.

$$\max(\frac{y}{z}, \frac{z}{y}) < \delta \tag{2.1}$$

For surface normal metrics, we use the metrics in Equation 2.2 where δ is 11.25, 22.5, and 30. $\hat{y}$ is the ground truth pixel-wise surface normal and $\hat{z}$ is the final output of the task decoder both normalized to a unit vector where each normal is also clipped between -1.0 and 1.0 in case of rounding errors. We also consider mean average error which is simply

the mean of the result of the left hand side of the equation (ignore δ) across all valid pixels.

$$\frac{180}{\pi} \arccos\left(\sum_{i=0}^{3} \hat{y}^{(i)} \hat{z}^{(i)}\right) < \delta \tag{2.2}$$

For semantic labeling metrics, we use the mean intersection over union shown in Equation 2.3. y is the ground truth pixel-wise depth and z is the final output of the task decoder. Occasionally, we will also use pixel accuracy which is just the mean accuracy of all of the pixels in an image. This is common and we will not discuss it at length here.

$$\text{mIoU} = \sum_{i=0}^{n} \frac{y_i \cap z_i}{y_i \cup z_i} \tag{2.3}$$

For unsupervised clustering, we use the cluster purity index, which calculates the percent of objects that were classified correctly. This is shown in Equation 2.4 where $\Omega = w_1, w_2, ...w_k$, $C = c_1, c_2...c_j$, N is the total number of samples, Ω is one set of clusters, and C is another set of clusters or labels. This calculates "correctly assigned" clusters.

$$\text{Purity}(\Omega, C) = \frac{1}{N} \sum_{i=1}^{k} \max_{j} |w_i \cap c_j| \tag{2.4}$$

For comparing multi-task to single-task, we introduce a metric called % improvement ($\%\tau$), which shows the average improvement each task had over the single task version. This is shown in Equation 2.5 where for each task i, s_i is the metric for the single task method and m_i is the metric for the multi-task method. The result is multiplied by 100 after subtracting 1 to convert it into percent better (positive) or worse (negative) than the single task results. For surface normals, we use the metric of δ is 11.25 in Equation 2.2, for depth we use δ is 1.25 in Equation 2.1, and for semantics, we use the mIoU defined in Equation 2.3.

$$\%\tau = 100\left(\frac{1}{n} \sum_{i=0}^{n} \frac{m_i}{s_i} - 1\right) \tag{2.5}$$

We also use a variant of this that accounts for the increase of the model size over a single task. We call this percent improvement per FLOP increase percentage (PIPFIP). It is simply ($\%_0\tau$) divided by the percent increase in model size given by multi-task model size f_m and the single label task size f_l in Equation 2.6.

$$\text{PIPFIP} = \frac{\%_0\tau}{100(\frac{f_m}{f_l} - 1)} \tag{2.6}$$

CHAPTER 3
RELATED WORK

In this chapter, we will discuss many works that inspired this dissertation as well as those related to it. The scope of 3D computer vision is incredibly large so this will be limited to those works we directly relate to and those we leverage information from. We are specifically interested in leveraging 3D information as context for dynamic scene understanding. By 3D, we mean some representation beyond RGB images that includes spatial components (depth, surface normals, etc.). When we say leveraging it as context, this means leveraging the 3D information to better a method's understanding of the world. For example, surface normals can give information about the shape of a room. i.e. floors are flat. By dynamic scene understanding, we mean understanding the surroundings from a camera in a dynamic environment, meaning a moving camera and moving objects.

3.1 Unsupervised Segmentation

One way we propose using 3D contextual information for scene understanding is through unsupervised segmentation. Unsupervised segmentation can group regions of related elements into regions that can be used for separating information or even as feature selection. First, we will discuss the background of unsupervised segmentation, which we propose a method for in Chapter 4.

3.1.1 Image Segmentation

Image segmentation and grouping has a large literature history going back to the 1970s. Our work is most inspired by the more recent Felzenszwalb and Huttenlocher image segmentation algorithm [18], which is a graph based image segmentation algorithm, which uses a modified Kruskal's algorithm on a minimum spanning tree and union-find to effi-

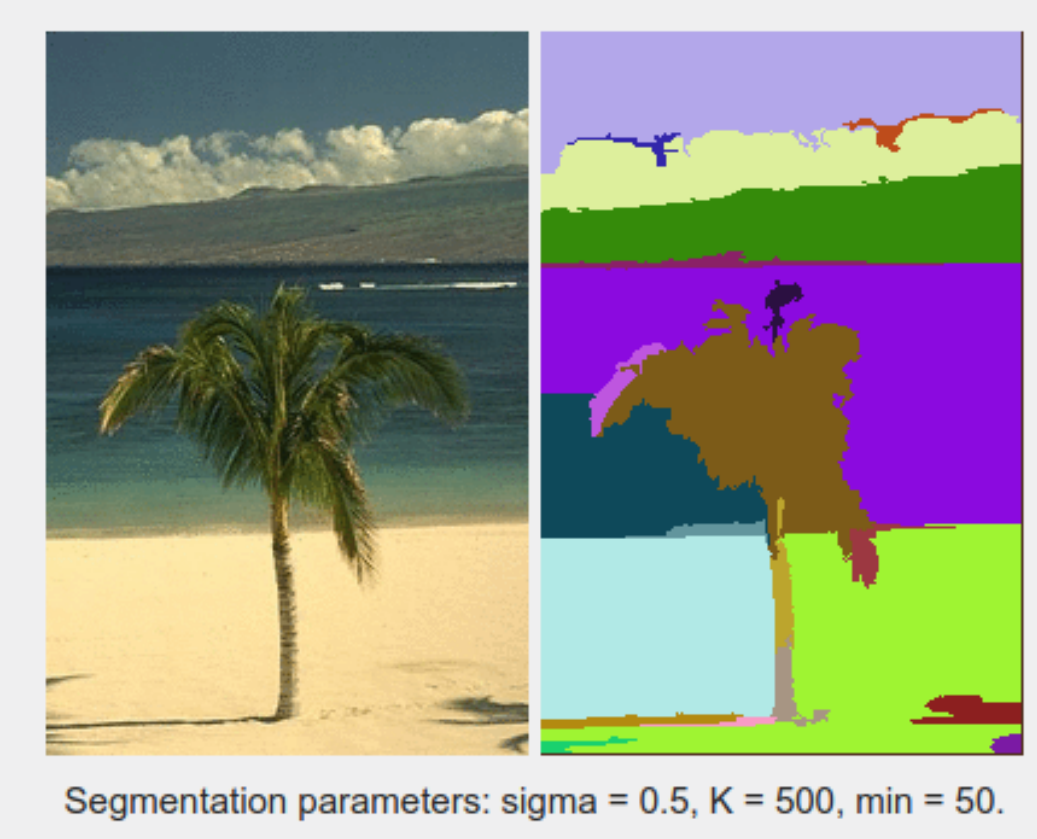


Figure 3.1: An example of segmentation from Felzenszwalb [18].

ciently merge pixels in a graph. Graph based segmentation technique tend to form a graph $G = (V, E)$, where V is all of the nodes in the graph (in this case pixels, but in other cases voxels or even toxels), and E are the edges between the nodes, generally found by some distance metric between neighboring pixels. An example from [18] is shown in Figure 3.1. They use 3 parameters, σ which is simply a Gaussian for smoothing the image beforehand, k which determines how far away the regions are before we stop merging them, and min, which enforces that no region is below a minimum size (in number of pixels) by merging with it's neighbor. This method of graph-based segmentation is very robust and has been extended in many ways for video, which we discuss below.

Figure 3.2: An example of segmentation from Grundmann [19].

3.1.2 Video Segmentation

One way to organize related work of unsupervised video segmentation is use the five types of algorithms for super-voxel video segmentation analyzed by Xu and Corso [20]. (A) Paris and Durand [21] propose a *mean shift* method that achieves hierarchical segmentation in videos using topological persistence using the classic mode-seeking mean-shift algorithm interpreted under Morse theory as a topological decomposition of the feature space. (B) [22] use *Nystrom normalized cuts*, in which the Nystrom approximation is applied to solve the normalized cut problem, for spatiotemporal grouping. (C) *Segmentation by Weighted Aggregation (SWA)* [23] is a variant of optimizing the normalized cut that computes a hierarchy of sequentially coarser segments by an algebraic multi-grid solver. (D) *Graph-Based (GB)* is an adaptation of the Felzenszwalb and Huttenlocher image segmentation algorithm [18] to video segmentation by building the graph in the spatiotemporal volume where voxels (volumetric pixels) are nodes connected to 26 neighbors. (E) *Hierarchical graph based (GBH)* is an algorithm for video segmentation proposed in [19] that iteratively builds a tree structure of region graphs, starting from over-segmented spatiotemporal volumes obtained using the method illustrated above. The regions are described by

LAB histograms of the voxel members, the edge weights are defined by the χ^2 distance, and the regions are merged using the same technique as in [18]. Grundmann's[19] segmentation method has similarly robust segmentation but now can continue to do so over time as shown in Figure 3.2.

3.1.3 RGBD Video Segmentation

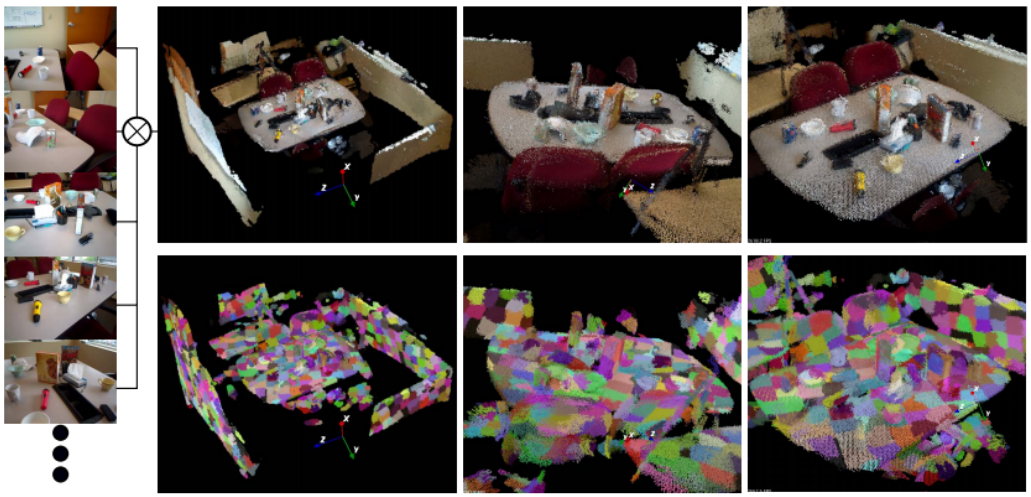


Figure 3.3: An example of segmentation from Papon [24].

Before our work in Chapter 4, there had been little work on unsupervised segmentation of streaming 3D point clouds besides [25], which uses a parallel Metropolis algorithm and the Potts model and focuses only on a specific environment and [26], which combines color and depth and uses spectral graph clustering without a hierarchical method. We were unable to compare against either of these since no code or data was released. Additionally, most of the research conducted in 3D segmentation of point clouds has been focused on static scenes, without considering time. Papon et al.[24] demonstrate a graph-based voxel segmentation method as an extension to what we have already discussed. An example from their paper is shown in Figure 3.3.

Other examples RGBD segmentation are planar and non-planar surfaces are segmented from 3D point clouds using either NURBS [27, 28], surface normal segmentation [29],

or 3D shape matching [30] to identify objects with a particular shape. Scene modeling is performed by analyzing support relationships of the regions [15] or contextual modeling of both super-pixel MRFs and paths in segmentation trees [31]. Temporal evolution of the 3D point cloud has been considered in cases where a learned 3D model of the segmented object is available, such as in the simultaneous segmentation and pose tracking approach of [32] or the segmentation-tracking method of an arbitrary untrained object in [33].

3.2 Unsupervised Object Clustering and Classification

As opposed to unsupervised segmentation which uses similarity metrics to merge regions of elements, some methods try to learn how to cluster and classify regions by using features. Though they are sometimes less robust than some of the unsupervised segmentation methods above, these learned methods can give more information about regions and be used to learn new to represent and classify new objects from large sets of unlabeled data such as videos. Unsupervised learning [34] and unsupervised deep learning [35, 36, 37, 38, 39] are central topics in machine learning applied to classification tasks. Many approaches have been proposed, leading to better clustering, dimensionality reduction, and embedding construction approaches. Building specialized, deep embeddings to help computer vision tasks is also a very active research area [40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50]. The main focus is on learning feature representations which can then be applied to transfer learning [41, 51, 45, 43]. These approaches have shown improved results on subsequent classification tasks, especially for small datasets or complicated high dimensional data such as video.

3.2.1 Unsupervised Object Retrieval

One sub-problem of unsupervised clustering that we look at in Chapter 5 is unsupervised object retrieval. These methods attempt to learn how to extract *important* objects from large amounts of data without labels (i.e. that is a car on a road and therefore something

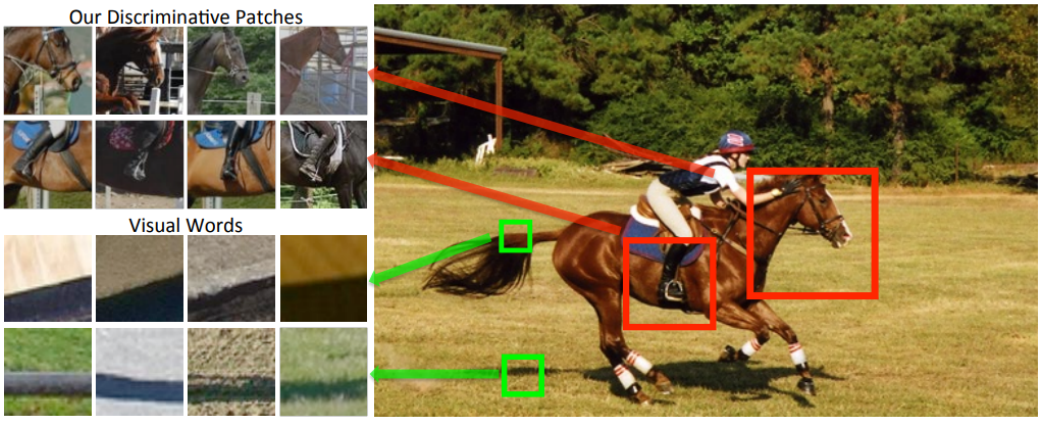


Figure 3.4: An example of a method for unsupervised discriminative patches using clustering from Singh [52].

important to learn about). One method of unsupervised visual object retrieval is object discovery from videos [53, 54, 55, 56]. This is a an emergent topic of research, exploiting the massive amounts of available (unannotated) video data, for example, using tracking as supervision [53]. This allows for an oracle system that can extract and label important objects without supervision. Unsupervised clustering of visual objects, given an initial feature representation, has also been a topic of large interest in computer vision with several foundational approaches proposed, e.g., topic models, spectral clustering [57, 58, 59]. Some methods such as that shown in Figure 3.4 attempt to learn discriminative patches without the need for video such as Singh et al. [52]. Deep clustering with jointly learning the embedding and the clustering is considered previously by various approaches [60, 61, 62, 63] but generally focus on small, cleaned datasets such as MNIST and Cifar.

3.2.2 Self Supervised Object Retrieval

Our work in Chapter 5 focuses on both unsupervised retrieval and discovery of visual object categories. It is aligned to prior object extraction work mentioned above, e.g. Singh et al. [52], whereas we here extract and classify full objects rather than re-occurring parts. To extract full objects, Pathak et al. [45] use optical flow as supervision in videos as shown

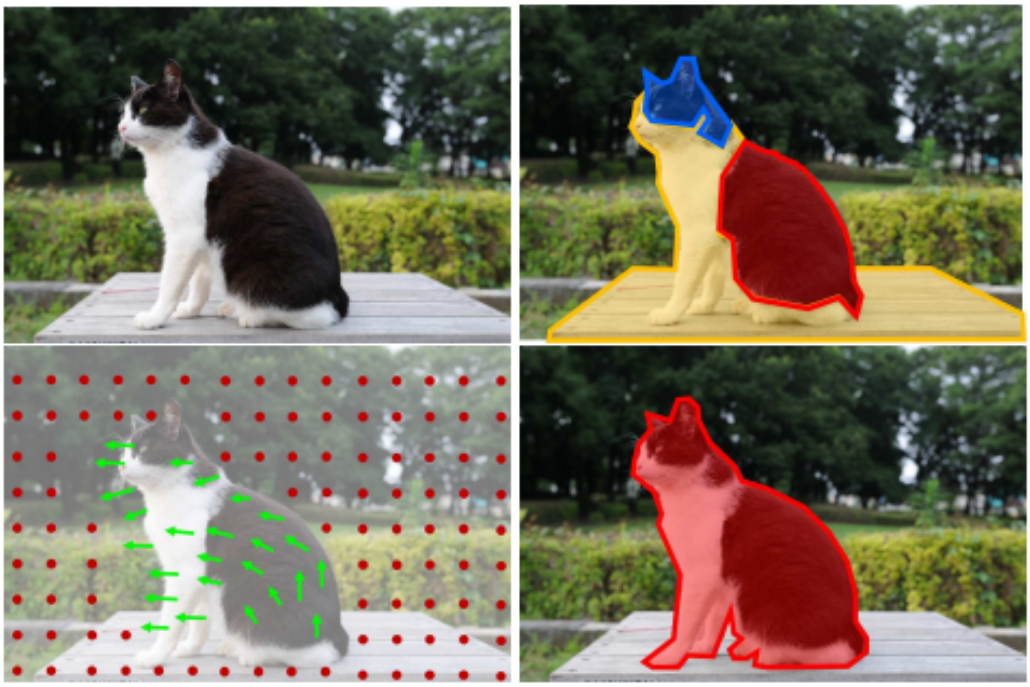


Figure 3.5: An example of a method for self-supervised clustering from video by Pathak [45].

in Figure 3.5. However, they use the features learned for transfer learning onto supervised tasks, whereas we obtain full semantic classes in unsupervised manner.

Semi-supervised or weakly supervised methods are also very common in many applications. For example Laplacian label propagation [64] and related techniques embed the dataset into a smaller dimensional space where labeled examples can propagate their labels to examples closer to them in space. Such techniques could be very useful for our approach too, as including a handful of manually labeled examples can provide more well defined clusters. This is also a practically feasible approach, as labeling a small number of examples is not particularly costly.

Another large body of work on learning object segmentation from videos is [65, 66], but they consider small datasets without much variability. In Chapter 5, we consider much more challenging outdoor scenes with much variability, such as urban environments.

3.3 Supervised Learning for Scene Understanding

Supervised learning has offered major breakthroughs in classification into multiple categories with thousands or millions of labeled examples (*e.g.*, [67, 68, 69, 70, 71] and many others). These methods rely on a lot of annotated data for training in order to perform well. While very foundational, fully supervised learning may not be always applicable or practical, as labeling is an inefficient and expensive process. Instead, learning from unlabeled data or weakly labeled data is a reality for many complex tasks. However, if enough labeled data is available, supervised methods tend to drastically outperform unsupervised or self-supervised methods.

3.3.1 Semantic Labeling

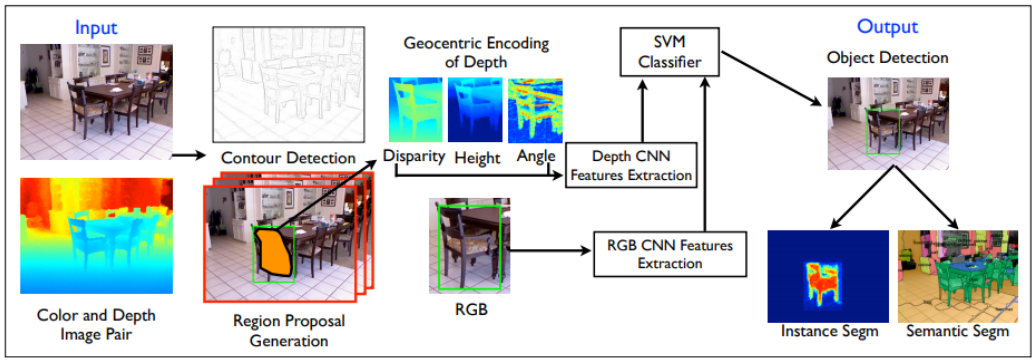


Figure 3.6: An example of a method for supervised semantic labeling from Gupta [72].

Most previous work has been focused on semantically labeling each pixel. Recently this has been extended to be each super pixel or patch or label using convolutional neural networks or conditional random fields. In the past, this was mostly done in outdoor datasets and on RGB images instead of 3D data. On 3D indoor scenes, most of the work has been focused on categorizing the environment and that work which does semantically label uses pixel or super pixels to train classifiers.

In [73], the authors focus on labeling scenes only (not objects or segments) such as

office, kitchen, hallway, etc; [74] and [75] label scenes the same way with an actual robot. Nuchter et al.[74] only label objects as the wall, floor, ceiling, or doors. These are classes that don't fit the challenging categorization we are trying to achieve. [76] generates meaningful and accurate semantic labeling; however, it trains specific objects (such as a bed or printer) and tests them assuming they are in the same environment. So although they can determine useful objects in an environment, they require knowledge of the current room. Richtsfeld et al.[28] use NURBS fitted patches to find graspable objects but do not classify them.

Silberman et al.[15] introduced the NYU dataset and a method to semantically label indoor scenes. They use a contour based method with structural inference and a set of various supervised learning features to train a SVM. As shown in Figure 3.6, Gupta et al.[77, 72] discuss using bounding box detectors for instance labeling and using an improvement of [15] with geometric encoding and an additive SVM kernel for semantic labeling. Couprie et al.[78] use a multi-scale convolutional neural network to learn unsupervised features and impose labels on super pixels. They also introduced a new subset of the NYU V2 dataset to test, which is based on the labels occurring most frequently in the dataset. This allows us to determine how an algorithm does labeling more specific categories. Wang et al.[79] also use an unsupervised method but instead use multi-modality learning in order to learn features that a linear SVM is trained on to generate semantically-labeled super pixels. Lin et al. [80] focus on recognizing cuboid objects by extending CPMC [81] to leverage 3D data and then use a conditional random field to generate semantic labels. Stuckler et al. [82] use temporal information to create an object-class segmentation using SLAM and a random decision forest in order to generate a fully labeled 3D map.

3.3.2 Surface Normal Estimation

Traditional methods for estimating surface normals were largely limited by sources for ground truth data, and instead incorporated explicit priors such as shading, vanishing points

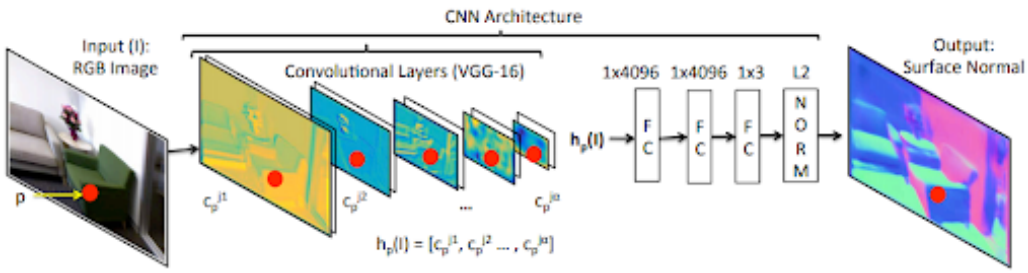


Figure 3.7: An example of a method for deep surface normal estimation from RGB images from Bansal [83].

[84], or world constraints [85]. With the advent of widely available and inexpensive depth sensors, data-driven approaches to this problem became more popular. Ladicky et al. [86] introduced a discriminatively trained learning based algorithm by combining pixel and segment-level cues. Fouhey [87] and colleagues explored the use of learned sparse 3D geometric primitives and higher level constraints to predict surface normals.

As discussed earlier, convolutional neural networks (CNNs) have proven to be a very effective means of tackling a wide range of image-level tasks. Wang et al.[88] introduced the first CNN-based method to solve the problem of dense surface normal estimation by fusing both scene level and patch level predictions. Eigen and colleagues [89] were the first to predict depth, surface normals, and semantic segmentation using the same multi-scale network architecture for each task (though not jointly). Bansal et al.[83] improved upon this architecture using skip connections and used it as input to jointly predict pose and style of 3D CAD models from an RGB image. This was one of the first deep encoder-decoder models for surface normal estimation from an RGB image and their model is shown in Figure 3.7. In SURGE, Wang et al.[90] uses a dense CRF to refine the output of their CNN model and demonstrate higher quality results on planar regions.

Some work, such as [91] have had success purely predicting planar regions instead of surface normals, though this limits the scope of the problem. Another promising avenue for gathering data for surface normal estimation is synthetic rendering. Zhang et al.[92] train their normal prediction network on a large dataset of rendered images and fine-tune

it on real data. Ren et al.[93] use an unsupervised domain adaptation method based on adversarial learning to transfer learned features from synthetic to real images. In this chapter, we use synthetic data in our training but batch-wise mix it with real data in an end to end training setup.

3.3.3 Depth Estimation

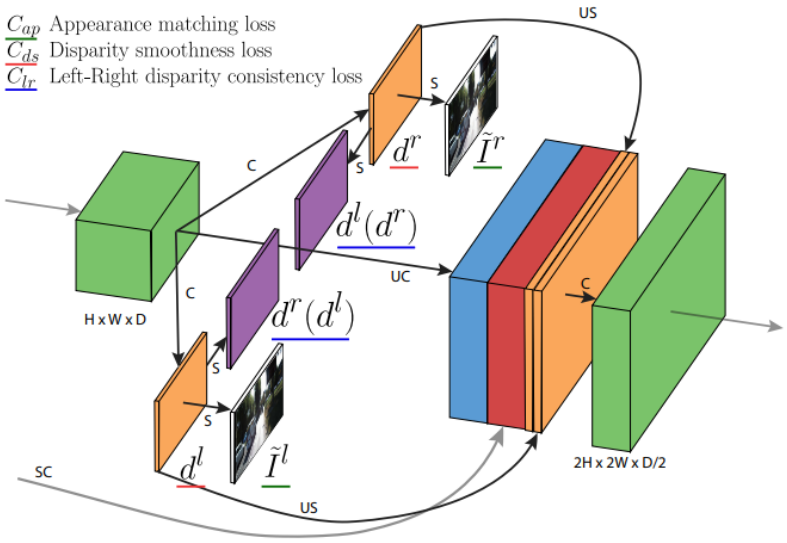


Figure 3.8: The loss module from Godard [94] using left and right disparity maps d^l and d^r. Here I^l and I^r are the left and right images, C=Convolution, UC=Up-Convolution, S=Bilinear sampling, US=Upsampling, and SC=Skip connection.

There is a large body of work on depth estimation from RGB images. This has only increased with the availability of the Kinect and other depth sensor devices. Some of this work focuses on using pairs of images such as [95]. Other work more recently focuses on depth estimation from only a single RGB image. Many of these methods focus on learning to predict depth with a convolutional neural network (CNN) such as Liu et al. [96]. Eigen [89] similarly predict depth from RGB images using a deep learning architecture. More recent works have focused on reconstruction losses, either from left-right image pairs [94] shown in Figure 3.8 or from camera motion with video [97, 98]. These methods attempt to learn depth and then use it to reconstruct from image time I_t to I_{t+1} or from a

left image to a right image in a stereo pair. This self supervision allows the reconstruction loss to be used to learn the depth.

3.4 Multi-Task Learning for Scene Understanding

Multi-task learning has a long history but most of this section will focus on recent techniques. Our approach is inspired by early work by Caruana et al.[99] who discuss using hard parameter sharing neural networks to predict several tasks. This methodology though being around since 1993, has not been leveraged fully with current encoder-decoder networks. There are many datasets used to leverage pixel-wise tasks (semantic labeling, depth, normals, HHA, curvature, etc), both synthetic like Scenenet [14] and real such as Scannet [16], NYUDv2 [15], and Matterport [100]. Availability of RGBD data has resulted on its use to predict semantic labels [72]. Different data source fusions have been developed (early, mid, and late) in order to determine where in a network to ”join” the information. The inverse, using only RGB to predict semantic labels, depth, normals, etc., has also become a topic of interest [89]. We briefly cover these efforts here.

Eigen et al. [89] use a common multi-scale fully convolutional network (FCN) architecture to predict depth, surface normals, and semantic labels but not jointly as shown in Figure 3.9. They only share the architecture layout but train separately. This was one of the first works on FCNs for joint tasks. Gupta et al.[72] later work on NYUDv2 [15] and introduce an encoding called HHA (horizontal disparity, height above ground, and angle). They then use that encoding to train a CNN whose features are combined with features from a CNN trained on RGB in order to do object detection and semantic segmentation (late fusion).

Other methods chose to combine networks or learn shared parameters with soft parameter sharing. Ruder et al. [102] discuss multi-task learning using a method they call sluice networks where parameters control which sub-spaces are shared between main and auxiliary tasks. There is a shared input layer and task-specific output layers and all other

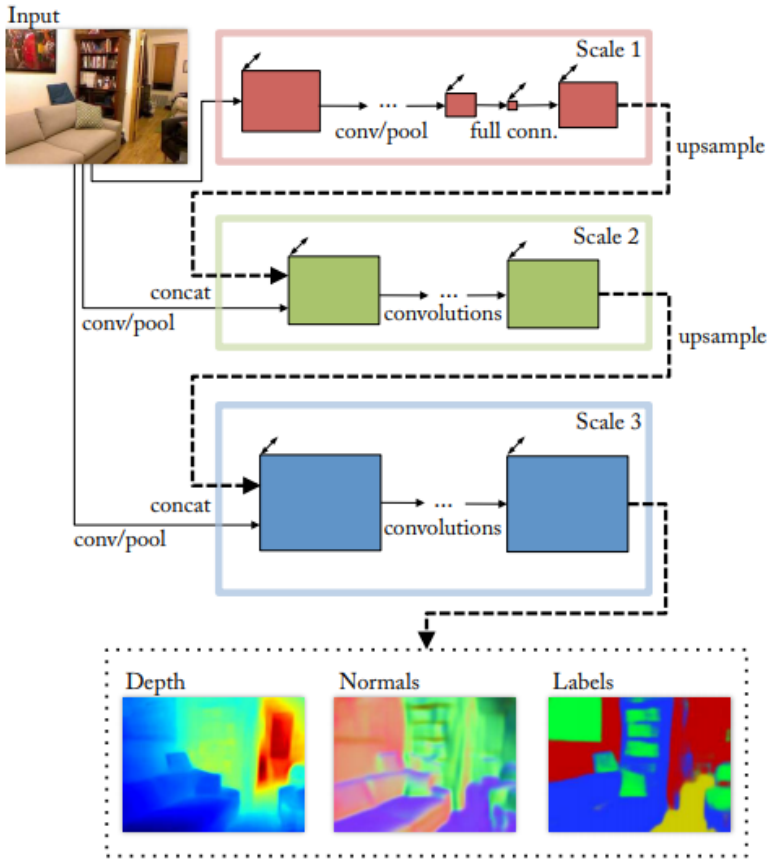


Figure 3.9: Eigen's method from [89] for a single network that can be adapted to do 3 different tasks (though not concurrently) showing that neural networks can learn different tasks individually without changing the architecture.

layers are shared with a parameter to control what is used for each prediction. They show small networks for NLP tasks as opposed to pixel-wise predictions which involve larger, more complex architectures with an encoder-decoder structure. Misra et al. [101] introduce cross-stitch networks with a similar method of [102] using a parameter to control the flow between the layers of the networks as shown in Figure 3.10. They try to learn how to stitch together two networks with a shared representation as a linear combination of activations. They explore when to split a network for two separate tasks but only from a FCN style network based on Alexnet [103]. This is different from more current methods which use those architectures as an encoder and have a separate decoder per task. Jafari et al. [104] use separately trained networks for depth and semantics as input to a joint refinement network

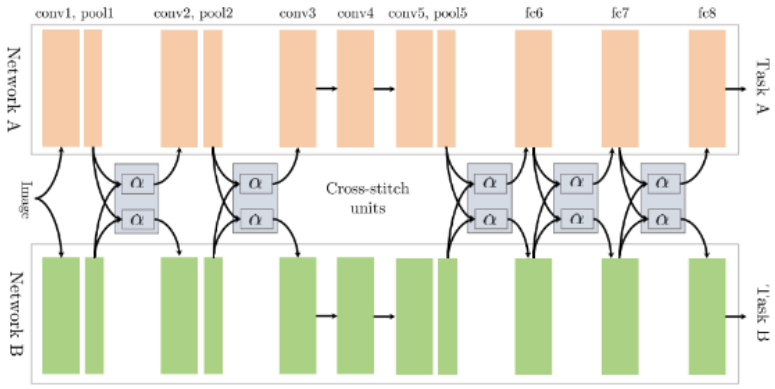


Figure 3.10: Misra's method from [101] for a soft-parameter sharing method that stitches two individual tasks together.

with the hope of mutually improving both results using cross-modality influences. Many of these soft parameter sharing methods can be transformed into others by specifically setting certain parameters. However, they can be difficult to learn and require a full network trained on each individual task which is a large amount of FLOPS.

Other hard parameter sharing work has focused on using several different modalities as both inputs and outputs or as intermediate tasks. Kuga et al. [105] use several different modalities as both inputs and outputs with each having it's own encoder and decoder but sharing skip connections and a shared latent representation. Xu et al. [106] use depth, surface normals, contours, and semantics as a set of intermediate tasks, which are then used via a multi-modal input into a distillation model. In this method, there is one shared encoder with each auxiliary task having a separate decoder. The results of that are then put into the multi-modal distillation module for final task prediction, where each task has it's own decoder. Maninis et al. [107] use a shared encoder with learned soft attention modules to train one network to do several tasks each one at a time, only focusing on the encoder.

3.4.1 3D Multi-Task Learning

Researchers have also begun to explore the connections between various 3D pixel level labeling tasks. Dharmasiri et al.[109] demonstrate that by having a single network jointly pre-

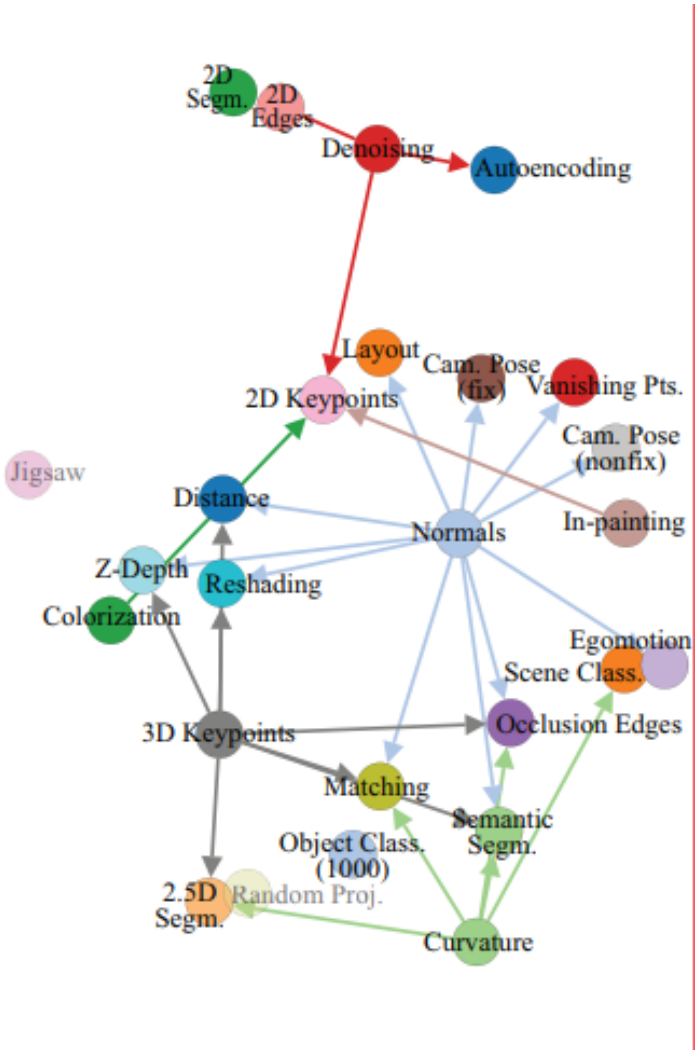


Figure 3.11: An example of a taskonomy from Zamir[108] for task-relatedness.

dict surface normals, depth, and curvature, they are able to almost match or improve upon networks tuned for these tasks independently. Similarly, Nekrasov [110] and colleagues explored the connections between depth estimation and semantic segmentation with a focus on the effects of asymmetric dataset sizes. Xu et al.[106] developed a prediction and distillation network that uses multiple intermediate representations such as contours and surface normals to achieve the final task of depth estimation and scene parsing. Similarly, Kokkinos [111] showed that a single unified architecture is capable of learning a wide range of image labeling tasks. A couple of works [112, 113] enforce consistency between joint

predictions of depth and normals. As opposed to learning tasks together, Taskonomy [108] attempts to learn task-relatedness by transferring learning between modalities instead of doing multi-task learning with a focus on indoor 3D environments. An example of a single taskonomy showing task "relatedness" is shown in Figure 3.11. Note how close normals are to many tasks compared to z-depth.

In Chapters 7 and 8, we show that semantic segmentation can improve normal prediction when predicting both jointly which results in even higher performance on planar regions.

3.4.2 Multi-Task Encoder Decoder Architectures

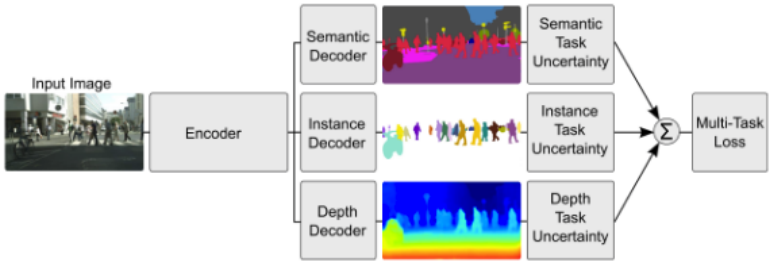


Figure 3.12: An example from Kendall [114] showing a multi-task encoder-decoder architecture using what we call *early fission* with a special multi-task loss.

Most work has focused on what we call the early fission model, where one shared encoder is used with separate decoders for each task. Each of these then requires clever methods in order to improve results. Initial works on many scene tasks including geometry and semantic labels [111] discuss task-interference and find that jointly learning can impede accuracy. [111] jointly handle many tasks end-to-end with a network architecture relying on task specific responses with many fusion layers with skip pooling across different resolutions to deal with different tasks. Their method also use memory-efficient back-propagation to handle training for many different tasks. Kendall et al. [114] use a shared encoder with separate decoders for each task and focus on using task-uncertainty to improve the training for each task as shown in Figure 3.12. Liu et al. [115] propose an

end-to-end multi-task attention network using one global feature pool and a soft attention module per task. This would be an example of late fission with an extra addition of the attention modules per task. Using this, they hope to learn task-shared and task-specific features in an automatic manner. Gao et al. [116] propose a fused network model by combining multiple single-task networks using discriminative dimensionality reduction with several 1x1 convolutions. This method however, is high in parameters as it combines two networks trained for separate tasks. In Chapter 7, we propose a method to predict surface normals and semantic labels with a real-time mobile model. After that, in Chapter 8, we go beyond previous work and try to analyze further output network fission schemes to allow for both task-shared features and task-independent features efficiently.

CHAPTER 4
EFFICIENT HIERARCHICAL GRAPH-BASED SEGMENTATION OF RGBD VIDEOS

In this chapter, we will discuss using depth and color to segment 3D scenes over time in an unsupervised manner. Here we use depth as a context to constrain and limit the color regions that we merge. We will show how the inclusion of depth improves these segmentation results and which scenarios it helps. We do this by simply utilizing one extra segmentation step, which maintains the real-time efficiency of image segmentation algorithms like that proposed by Felzenschwab[18].

4.1 Introduction

Segmentation is used as a building block for solving problems such as object detection, activity recognition, and scene understanding. Traditionally, segmentation involves finding RGB pixels with similar perceptual appearance in a *single image* and grouping them as discussed in Chapter 3 Section 3.1.1. As processing power has improved, there has been increasing emphasis upon *video segmentation*, in which pixels with similar appearance and spatiotemporal continuity are grouped together over a video volume[21, 19, 20]. Given the widespread availability of depth data from low-cost RGBD sensors, we are interested in robust unsupervised segmentation of *streaming 3D point clouds* with temporal coherence. Because we are using point clouds here, we can use shape as an additional cue to improve our segmentation as mentioned in the Thesis Statement (1.1).

In this work, we modify traditional segmentation by using depth as a constraint so that we only find RGB pixels with similar appearance within individual objects and not spanning across depth boundaries. This differs from previous works because we leverage the 3D context that color regions should never merge across depth boundaries.

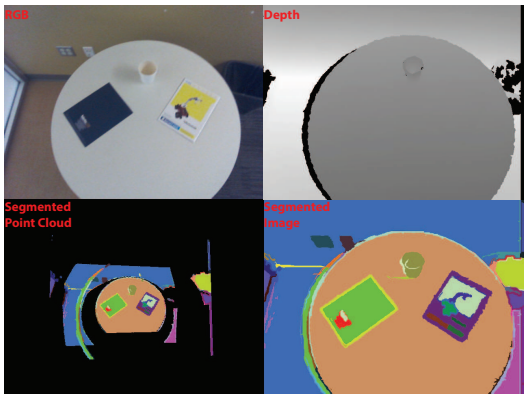


Figure 4.1: Segmentation of streaming 3D Point clouds. Top: RGB and depth maps used as input in our algorithm. Bottom: Rendered version of the point cloud and its frame segmentation output.

Our algorithm is aimed at *super-toxel* extraction; since voxels are limited to 3D data, we define a new term *toxels*. Toxels are *temporal voxels*, which can be understood as a generalization of voxels for 4D hyper-rectangles. Each toxel is a hyper-cube discretization of a continuous 4D spatiotemporal hyper-rectangle. Voxels are generally used to refer to the discretization of 3D volumes or the discretization of 2D frames over time. When these are combined, we get XYZT data, which can be discretized using toxels. Super-toxels are just large groupings of toxels. This allows us to respect both depth and color as important cues.

Our method uses measurements such as color, spatial coordinates, and RGBD optical flow to build a hierarchical region tree for each sequence of n frames (we use $n = 8$), which are then sequentially matched to produce long-term continuity of region identities throughout the sequence. This bottom-up design avoids limitations on the length of the video or on the amount of memory needed due to the high volume of data. In contrast to segmentation-tracking methods, we do not assume *a priori* knowledge of the content of the scene and/or objects, and we do not perform initialization of seeds or any other type of human intervention. Our motivation is that regions should not merge across depth boundaries despite color similarity. Our method was tested on several challenging sequences from the TUM

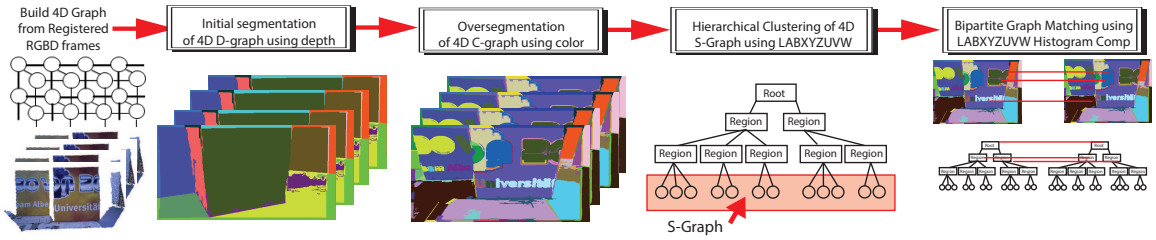


Figure 4.2: Schematic of our method to segment streaming 3D point clouds.

Dataset [117], the NYUDv2 Dataset [15], and our own examples. It produces meaningful segmentation, even under difficult conditions such as significant motion of scene objects, camera motion, illumination changes, and partial occlusion.

Our primary contributions in this chapter are: (1) A robust and scalable, RGBD video segmentation framework for streaming 3D data. (2) A streaming method that maintains temporal consistency and can be run on a robot or on videos of indefinite length. (3) An efficient framework that runs at 0.8 fps but can be downsized to run pairwise in near real-time at 15 fps. (4) A nonlinear multistage method to segment color and depth that used 3D context to enforce that regions never merge over depth boundaries despite color similarities. And finally (5) The ability to segment and maintain temporal coherence with camera movement and object movement using these 3D cues to improve results. The code and data are publicly available on the project web page[118].

4.2 Method

Our approach involves four main steps, as illustrated in Figure 4.2. First, a segmentation of 8 consecutive frames using the depth and motion information is performed. In the second step, an over-segmentation of the frames is done using color and motion information while respecting depth boundaries. Next, histograms of the resulting regions are used to build a hierarchical segmentation of the spatiotemporal volume represented as a dendrogram, which can then yield a particular segmentation depending on the desired segmentation level output. The final step performs a bipartite graph matching with the 8 previous frames

with 4 frames overlapping to enforce the consistency of region identities over time.

4.2.1 Spatiotemporal segmentation

Graph-based segmentation: A natural approach to segmenting RGBD images would be to simply use [18] after setting the edge weights to a weighted combination of differences in depth and color: $(1-\alpha)d_C(p_1,p_2)+\alpha d_D(p_1,p_2)$, where p_1 and p_2 are two pixels, d_C is the difference in color, d_D is the difference in depth, and α is a scalar weighting the relative importance between the two. We shall show in the experimental results in Figure 4.3 that this approach does not yield good results in practice, because there is no value of α that will work consistently well for a variety of images. In fact, in many cases no value of α works well even for the various regions of a single image. We find that just blindly combining depth and color does not work well; it needs to be done intentionally by using 3D context of object boundaries as a constraint. To do this, we use the multistage approach described in Section 4.2.1.

RGBD optical flow: We compute optical flow between each consecutive pair of RGBD images. First, the depth values are converted to XYZ coordinates in 3D space by using the internal and external calibration parameters of the visible and IR cameras on the sensor to register the RGB and depth images. Then we compute dense optical flow between the two RGB images. Previous extensions of optical flow to 3D [119] assume a continuous spatiotemporal volume of intensity values on which differentiation is then performed. In our case, however, the depth values are already known, thus enabling a much simpler calculation. We compute 2D optical flow $(\Delta i, \Delta j)$ using a consecutive pair of RGB images based on Farneback's method [120]. Therefore, since the RGB and depth images are registered, for the value $D_{n-1}(i,j)$ in the depth map at location (i,j) at time n, there exists a corresponding depth value $D_n(i+\Delta i, j+\Delta j)$ in the next frame. The optical flow, w,

along the z axis, is then simply the difference between the two depths:

$$w = D_n(i + \Delta i, j + \Delta j) - D_{n-1}(i, j). \tag{4.1}$$

We also project the motion of the (i, j) pixels into 3D space using the camera calibration parameters.

Hence, the RGBD scene flow can be solved by extending Equation 4.1 and the calibration of the data and is defined as:

$$(u, v, w) = ((i_n - i_{n-1})\frac{z_n}{F_x}, (j_n - j_{n-1})\frac{z_n}{F_y}, D_n(i + \Delta i, j + \Delta j) - D_{n-1}(i, j)). \tag{4.2}$$

Segmentation using depth and color

Our approach relies on the following observation: If the scene consists only of convex objects, then every depth discontinuity corresponds to an object boundary. This observation was motivated by a study on primates [121] which concluded that depth and color perception are handled by separate channels in our nervous system that function quite distinctly. Obviously the world does not consist only of convex objects, but nevertheless we have found this observation to have great practical utility in everyday scenes. We exploit this observation by first segmenting based on depth alone, then based on color while preventing any merging to occur across depth discontinuities. We show in the experimental results that this approach yields more accurate segmentation than combining depth and color into a single graph.

We build a graph (called the D-graph) shown in Figure 4.2, in which each node corresponds to a toxel. Within each graph, nodes are connected to their 26-neighbors with edges whose weights are the absolute difference in the depth of the two toxels: $|D(x, y, z, t) - D(x', y', z', t')|$, where $D(x, y, z, t)$ is a function that yields the depth values from the spatiotemporal volume, and $(x', y', z', t') \in \mathcal{N}(x, y, z, t)$, where $\mathcal{N}(x, y, z, t)$ is the neighbor-

hood of toxel (x, y, z, t). Edges are then constructed between each frame $D(x, y, z, t-1)$ and $D(x, y, z, t)$ by connecting toxels based on the computed optical flow. Toxel $(x, y, z, t-1)$ in D is connected with toxel $(x+u, y+v, z+w, t)$ in D with an edge whose weight is given by $|D(x, y, z, t-1) - D(x+u, y+v, z+w, t)|$. The internal difference of a region $Int(R)$ is set as the highest edge weight in the minimum spanning tree of region R, and regions are merged according to [18]. The constant that determines the size of depth segments is called k_{depth}.

After the graph-based algorithm produces a segmentation according to the depth, another graph (called the C-graph) is created with the same nodes as before shown in Figure 4.2. As before, edges are created for each of the 26-neighbors, and edges between frames are set according to the RGBD optical flow vectors to be $|C(x, y, z, t-1) - C(x+u, y+v, z+w, t)|$ where $C(x, y, z, t)$ is a function that returns the euclidean difference in color. We use CIE LAB color space to ensure perceptual uniformity of color differences. Invoking the observation above, the edge weights connecting toxels that belong to different regions after processing the D-graph are set to infinity in the C-graph to forbid merging across depth discontinuities. The constant that determines the size of color segments is called k_{color}. After running the graph-based algorithm on the C-graph, we have an over-segmentation consisting of super-toxel regions across the spatiotemporal hyper-rectangle.

4.2.2 Hierarchical Processing

Once the over-segmentation has been performed, feature vectors are computed for each super-toxel region. For feature vectors, we use histograms of color, 3D position, and 3D optical flow, called LABXYZUVW histograms, using all the toxels in a region. For computational efficiency and storage requirements, rather than computing a single 9-dimensional histogram, we compute nine 1-dimensional histograms. Experimentally we find that 20 bins in each of the LAB and UVW features and 30 bins in each of the XYZ features is adequate to balance generalization with discriminatory power.

Using these feature vectors, a third graph (called the S-graph) is created to represent the regions as shown in Figure 4.2. Each node of the graph is a region computed by the super-toxel segmentation in Section 4.2.1 and edges are formed to join regions that are adjacent to each other (meaning that at least one toxel in one region is a 26-neighbor of at least one toxel in the other region). The weight of each edge is the difference between the two feature vectors, for which we use Equation 4.6. On this graph, instead of running the graph-based algorithm of [18], we use Kruskal's algorithm to compute the minimum spanning tree of the entire set of regions. The result of this algorithm is a dendrogram that captures a hierarchical representation of the merging in the form of a tree, in which the root node corresponds to the merging of the entire set.

By selecting a different threshold, a different cut in this minimum spanning tree can be found. To make the threshold parameter intuitive, we choose the percentage of regions to merge, which we denote by ζ. The value determines the coarseness of the produced results, with $\zeta = 0$ indicating that all the regions are retained from the previous step, whereas $\zeta = 100\%$ indicates that all the regions are merged into one. For values between these two extremes, the dendrogram is recursively traversed, accumulating all nodes until the desired percentage is obtained. Although the value ζ can be changed on-the-fly for any given image without incurring much computation to recompute the segments, we generally set $\zeta = 65\%$ for the entire sequence to avoid having to store all dendrograms for all frames.

4.2.3 Bipartite Graph Matching

After thresholding the dendrogram of the current pair of frames according to ζ, correspondence must be established between this thresholded dendrogram and that of the previous pair of frames. To achieve this correspondence, we perform bipartite graph matching between the two thresholded dendrograms using the stable marriage solution [122]. Figure 4.2 depicts the bipartite graph matching process, which considers the difference in region sizes, the change in location of the region centroid (after applying the 3D optical

flow), and the difference between the LABXYZUVW histograms.

Data is treated as two lists from the tree cut. The histogram match from each region's perspective is found with the histogram SAD difference equation between Region R and Region S being:

$$\Delta H = \sum_{i=1}^{\text{NUMBINS}} \sum_{\beta \in (l,a,b,x,y,z,u,v,w)} |\frac{R_\beta[i]}{R_N} - \frac{S_\beta[i]}{S_N}| \tag{4.3}$$

where R_N is the number of toxels in Region R and S_N is the number of toxels in Region S.

For each histogram, we compute the distance the centroid has traveled over time and the size difference.

The difference the centroid has traveled is computed as the SAD of the center points normalized by the size:

$$\Delta d = \frac{|R_x - S_x| + |R_y - S_y| + |R_z - S_z|}{R_N} \tag{4.4}$$

and the size difference is computed as the absolute value of the difference:

$$\Delta N = |R_N - S_N| \tag{4.5}$$

Starting with the closest histogram, we merge the regions R and S iff they each consider each other the best choice using the weight as defined by Equation 4.6:

$$h = \beta \Delta H + \gamma \Delta d + \epsilon \Delta N \tag{4.6}$$

where β, γ, and ϵ are fixed constraints to prevent over-merging, found by computing statistics on multiple sequences and determining 3σ for each constraint.

4.3 Experimental Results

Our algorithm was evaluated with several 3D RGBD sequences, some from the NYU Dataset [15], some from the TUM Dataset [117], and some that we obtained. To measure the accuracy of the algorithm, we annotated some images by hand to provide ground truth as 2D object boundaries. To account for slight inaccuracies in the position error of a boundary point, we use the Chamfer distance [123]. The boundary edges are extracted from the output segmentation images as well as of the ground truth images. For each pixel in the contour of the query image, we find the distance to the closest contour pixel in the ground-truth image (which is efficiently computed using the Chamfer distance), and then we sum these distances and normalize by dividing by the number of pixels in the image. The error metric is shown in Equation 4.7.

$$E_{bound} = \frac{\sum_i \|NN^1(\text{Out}_i - \text{GroundTruth})\|}{\text{Width} * \text{Height}}, \tag{4.7}$$

Where NN^1 represents the closest point of the boundary output Out_i to the ground truth.

As mentioned earlier, we were unable to find a value for α that enables the traditional graph-based algorithm in [18] to produce robust, accurate results across all sequences. This is demonstrated in Figure 4.3, where the output for various values of α are shown, none of which clearly delineates the objects properly. The linear combination never properly segments the table, table legs, cup, magazine, and magazine colored features in Figure 4.3, whereas our over-segmentation does. In other words, the multistage segmentation combination of first depth and then color segmentation provides better results than any linear combination of color and depth features as the primate studies from [121] suggested.

To quantify these results, we evaluated the output of the graph-based algorithm of [18] for different values of α for all three sequences (shown in Figures 4.6-4.8 corresponding to S1-S3). As shown in Figure 4.4, better results are obtained when color is weighted higher than depth (α is small). Nevertheless, no value of α yields an acceptable segmentation.

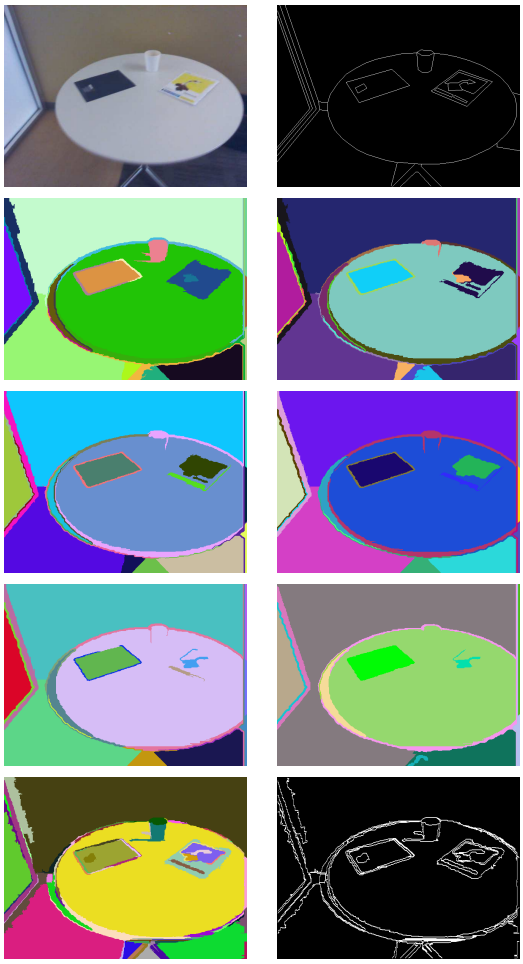


Figure 4.3: Top row: original RGB image (depth image not shown), and ground truth object boundaries. Middle three rows: The output of the graph-based algorithm using a linear combination of color and depth, with α values of 0.1, 0.2, 0.4, 0.6, 0.8, and 0.9; none produces a correct result. Bottom row: The output of the initial over-segmentation step of our algorithm (left), as well as the boundaries of this output (right).

Also shown is the initial over-segmentation step of our method. Since our method is independent of α, these results are plotted as horizontal lines. Note that the segmentation error of our multistage segmentation algorithm is lower than the output from the linear combination approach for any value of α.

Even though our algorithm does not depend on α, it does initially depend on k_{depth} and k_{color}. The left of Figure 4.4 shows the error (measured by Equation 4.7 where k is the parameter from [18]) obtained from segmenting just based on depth as the parameter k_{depth} is changed. Note that on all three images the output gets worse as this value is increased

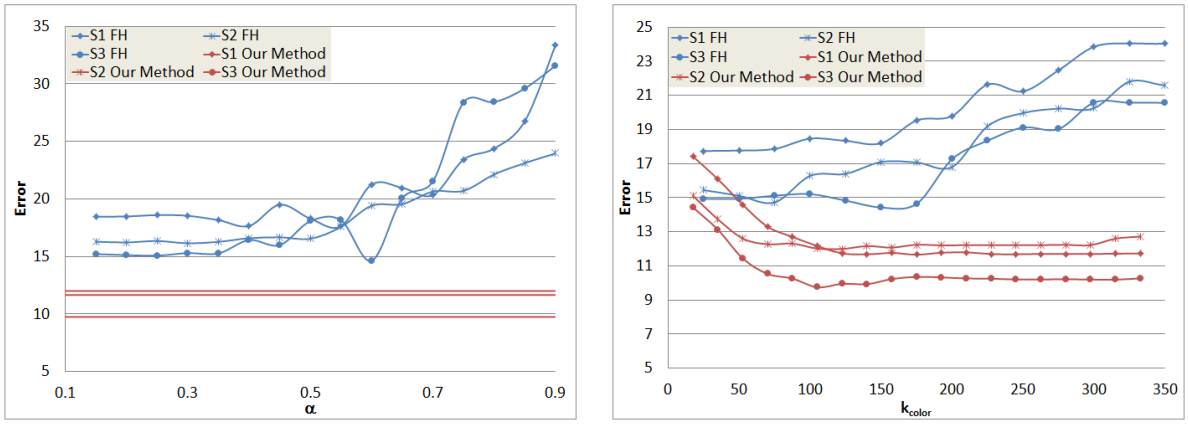


Figure 4.4: Left: The error based on the choice of α on the output of the linear combination algorithm on three sequences (blue lines), as well as the output of our hierarchical algorithm (red lines), which is not dependent on α Right: Effect of varying k_{color} on the output of our initial over-segmentation step, as well as the value of k on the output of [18] (with the best α possible.

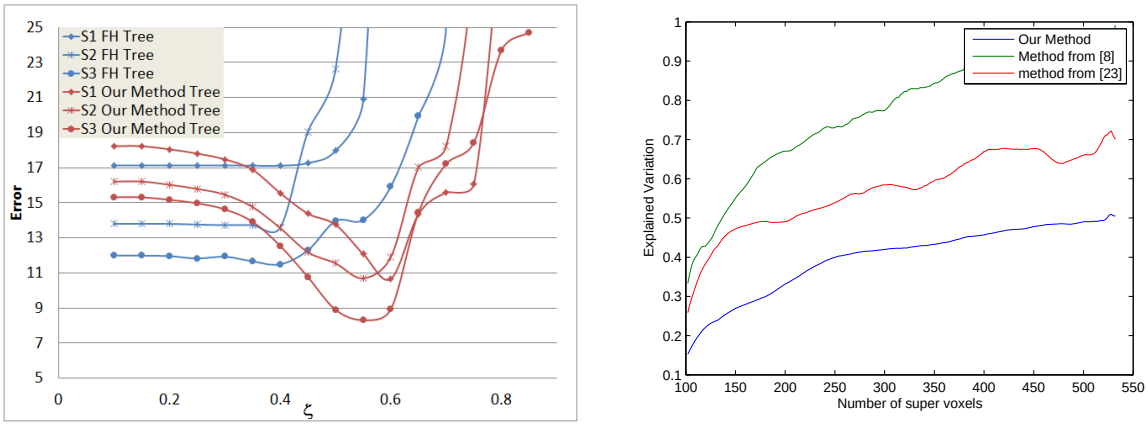


Figure 4.5: Left: Effect of varying ζ on the output of our hierarchical clustering applied to the output of either our initial over-segmentation step or that of [18] (with the best α and k possible). Right: The explained variation from [124] of our method, [124], and [19] on Sequence 2.

due to under-segmentation. On the right is a plot of all three sequences for both approaches discussed in 4.3, where our algorithm includes only the initial over-segmentation step. With a low k_{depth}, these plots show the effect of changing the value of k_{color} for our algorithm, or similarly of k for [18]. For nearly all values our method achieves less error.

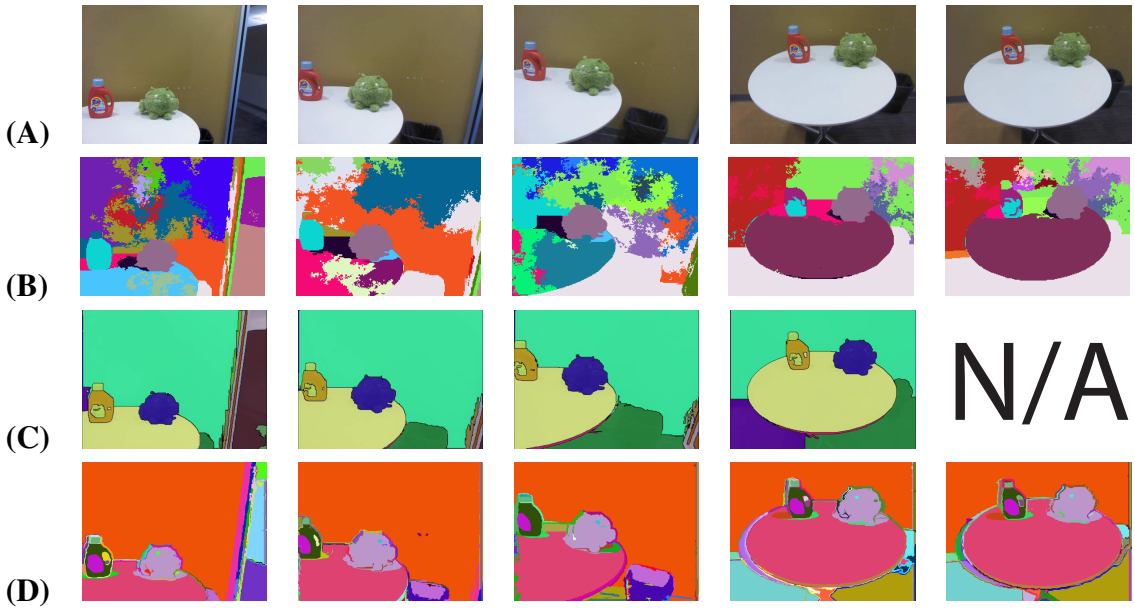


Figure 4.6: Sequence 1, (A) : RGB image, (B): Results from [124], (C): Results from [19] (N/A is used when the frame is beyond the length it can run), (D): Our Results

Figure 4.5 shows the error using the hierarchical method. On the left is a set of plots of all three sequences for both approaches, where we vary the hierarchical tree percentage cut. It can be seen that the hierarchy makes us invariant to the values of k_{depth} and k_{color} just as in [19]. On the right are plots of the error of our hierarchical step applied to the output of both our algorithm (initial over-segmentation step) and the algorithm of [18]. These plots answer two questions, namely, whether the hierarchical step improves results, and whether our initial over-segmentation step is needed (or whether the hierarchical step would work just as well if we simply used the algorithm of [18]). The answer to both questions is in the affirmative. The rightmost figure shows that our method also has a lower explained variation.

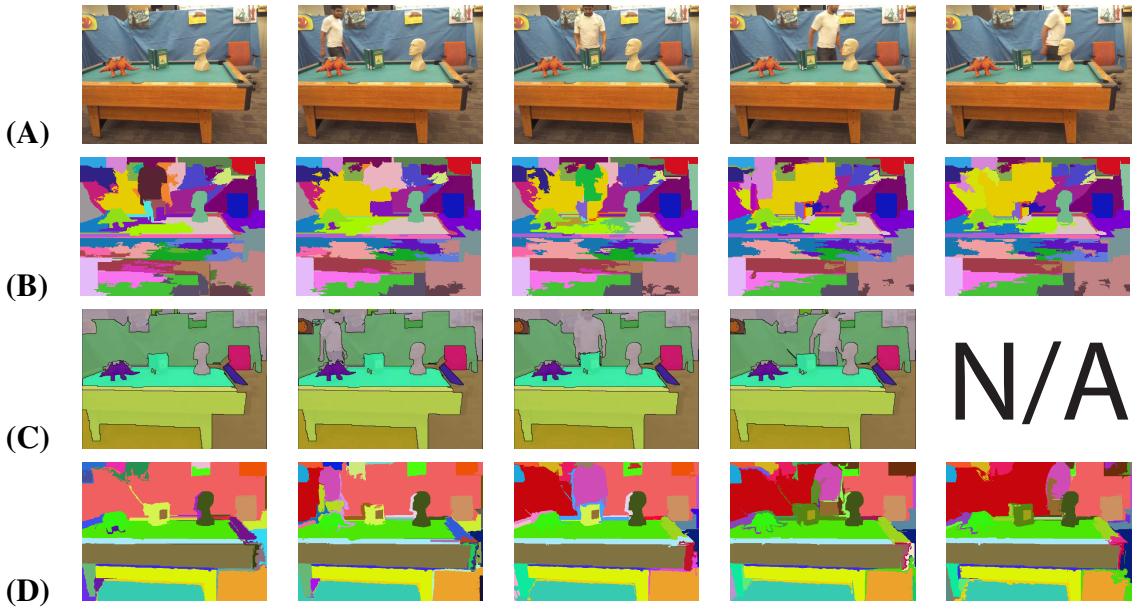


Figure 4.7: Sequence 2, (A) : RGB image, (B): Results from [124], (C): Results from [19] (N/A is used when the frame is beyond the length it can run), (D): Our Results

4.4 Conclusion

In this chapter, we have presented an unsupervised hierarchical segmentation algorithm for RGBD videos leveraging both shape and color by using 3D context to constrain segmentation. The approach exhibits accurate performance in terms of segments quality, temporal region identity coherence, and computational time. The algorithm uses a novel approach for combining the depth and color information to generate over-segmented regions in a sequence of frames. A hierarchical tree merges the resulting regions to a level defined by the user, and a bipartite matching algorithm is used to ensure temporal continuity. We showed that our method outperforms a linear combination of color and depth. We performed comparison against different graph segmentation combinations showing lower errors in terms of quality of the segmentation and thoroughly analyzing the effects of various parameters.

There are occasional problems with noise from the depth image and maintaining temporal consistency. This produces an excellent segmentation algorithm that respects both color and depth that we can use for further scene understanding tasks. As discussed in

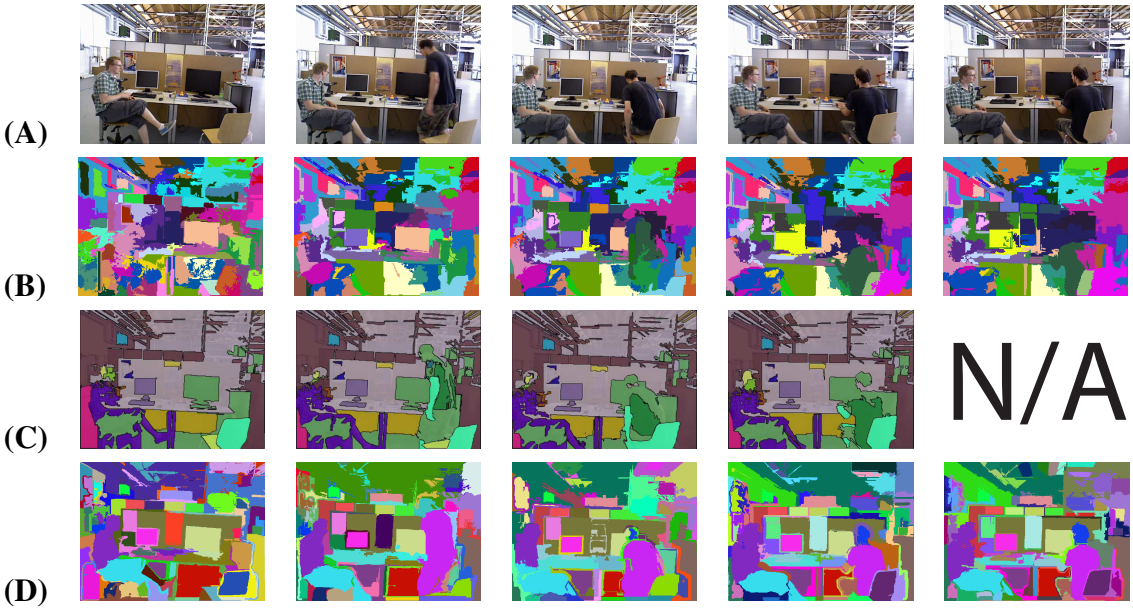


Figure 4.8: Sequence 3, (A) : RGB image, (B): Results from [124], (C): Results from [19] (N/A is used when the frame is beyond the length it can run), (D): Our Results

Chapter 3 Section 3.1.1, these segmentation regions can be used as features for other vision tasks such as classification and clustering. In Chapter's 5 and 6, we will use this method and extend on it to explore supervised semantic labeling with unsupervised segmentation and unsupervised clustering using both shape and color as cues.

CHAPTER 5

SELF-SUPERVISED DEEP CLUSTERING OF OBJECTS

In this chapter, we continue discussing using 3D contextual cues for unsupervised learning. In the previous chapter, we used depth and RGB as inputs for a graph-based segmentation algorithm. Here, we build off of that by using that contextual information from unlabeled videos of city street data to learn to retrieve objects that are semantically meaningful. We then attempt to cluster those objects without labels. We use a variant of the segmentation method from Chapter 4 to segment out meaningful objects from scenes and then learn to cluster them. Here the segments from the previous chapter are treated as meaningful proposals and we then learn whether they are meaningful and how to cluster them. In this work, the 3D context is both the known shape of the scene and the constrained environment we use (i.e. cars on a road).

5.1 Introduction

Much of human knowledge and learning is obtained by unsupervised observations [125]. Here we pose the question: *How much semantic knowledge can be retrieved from passively observing videos?* This work aims to learn about the existence of objects by only observing the world while moving in a scene[1]. Using depth and shape as a context is key here to understand what proposals to use in order to make them meaningful and useful. Depth is used to consider different objects in different manners and the plane of the ground is estimated from the depth in order to only consider objects on top of the road but below the sky. We then can leverage the segmentation from Chapter 4 to segment the remainder of the scene for object proposals to consider.

We show that semantically meaningful object proposals and categories can be learned

[1]Moving in the scene is required during training only, but testing is done from still images.

Figure 5.1: An overview of the method: self-supervision is used to learn proposals for full objects; unsupervised deep clustering is then used to uncover semantic object classes.

from video, with no manual supervision. This is a very challenging task as no labels for objects classes are available, *i.e.* there is no exemplar of what a car or a person should look like. Furthermore, an often overlooked but key difficulty in unsupervised visual learning is that labeled data also provides a very strong and powerful supervision of what constitutes a *full* object. We address both of these challenges here.

To obtain full object segments, we utilize unsupervised learning of depth from video [97, 94] and propose to cluster object components in the obtained 3D RGB space. We note that clustering in this space is advantageous as objects in 3D are much more likely to be well separated, despite inherent noise, and are naturally normalized for scale (*i.e.*, in 3D all cars should be of similar size, which is not true in image space). We see in our results that this approach is capable of retrieving full objects, and can also separate them into individual instances. Crucially, this process can be done without supervision since depth and ego-motion is obtained by self-supervised learning [97]. In contrast, prior algorithms for objectness or for proposal generation either require manual supervision to determine objectness [126], or provide a large number of proposals, the majority of which do not constitute full objects [127].

Given a set of objects proposals obtained by clustering in 3D RGB space, we now have important supervision of which object is foreground vs. background. We cluster them into classes while simultaneously learning the most suitable feature embedding. Building embeddings in unsupervised manner have shown to produce better feature representations for pre-training [45, 41]. We go a step further and, in addition to learning the representation, we learn unsupervised clustering at the same time, thus obtaining direct retrieval of full

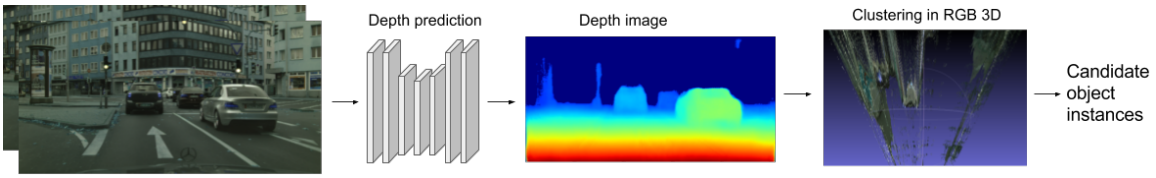


Figure 5.2: Schematic architecture which generates object proposals with self-supervised methods (e.g. learning of depth from video). The depth prediction convolutional stack is based on VGG [69]

semantic categories without supervision. To that end, we propose an unsupervised method that separates the proposals of potential objects into semantic classes using a fully differentiable memory-based deep clustering approach. The main idea is to store the potential cluster means as *weights* in a neural network at the higher levels of feature representation. This allows them to be *learned* jointly with the potential feature representation. Doing this jointly has the advantage that gradient descent can not only learn good weights for clustering, but it can also simultaneously change the embedding to allow for better clustering without the use of labels. We refer to this clustering as Unsupervised Memory Clustering (UMC).

5.2 Approach

5.2.1 Self-supervised proposal generation

The goal of the first step is to extract a set of object candidates from video frames. This is clearly a very challenging task and traditionally methods that do this require object bounding box supervision [71], whereas the unsupervised approaches, generate hundreds or thousands of candidates, many of which are partial objects [127]. We propose to extract proposals for *full* objects. The key idea is to first densely estimate depth, followed by clustering in RGBD space (Figure 5.2).

Unsupervised depth prediction from a single image. Let us start with estimating the depth of the scene per input image. Given a sequence of RGB images, Zhou et al. [97] showed that it is possible to learn to predict the depth per image and the ego-motion be-

tween two images, in an unsupervised way from monocular video. At inference time a single image input is sufficient to predict depth. During training, at least two consecutive images are needed [97, 94]. In our particular implementation, we used the algorithm [94] (open source implementation), which uses stereo images as inputs to impose additional consistencies, but these are needed for training only and not testing. During testing, either of the above-mentioned approaches (of Zhou et al. or of Godard et al.) needs a single image and both approaches have shown to be top performers in their respective categories. We consider this part of our approach as *self-supervised* due to the 3D constraints imposed as supervision which may come from the same or pair of images.

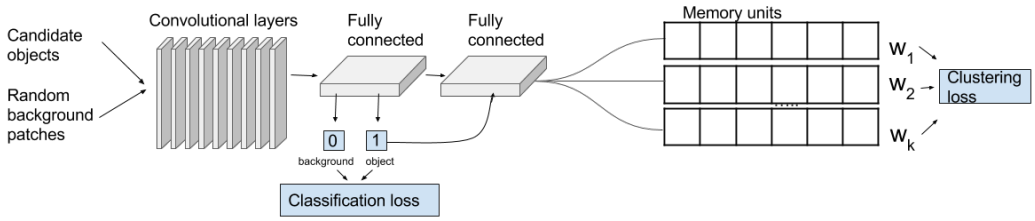


Figure 5.3: Unsupervised Memory Clustering with the clustering objective. Clusters will be learned as additional weights, jointly with the feature embedding for positive samples.

Clustering in RGB 3D In order to obtain object proposals, we use unsupervised segmentation and grouping to extract foreground objects from Chapter 4. Depth is estimated from the method described above (based on [94]). We extract the ground plane, using the Point Cloud Library (PCL) ground plane estimation method[128]. Points that are too high or far away are ignored. In order to segment well, we consider RGB, depth, and time. Segmentation is done by continuously grouping similar regions (or points), resulting in an initial over-segmentation [18]. These regions are then merged to yield a dendrogram using agglomerative clustering via a minimum spanning tree algorithm. A hierarchical merging method is then applied in order to merge regions which are similar (Figure 5.7 shows generated proposals).

5.2.2 Unsupervised memory clustering (UMC)

In this section, we address the problem of discovering semantic classes in an unsupervised way, given object proposals.

We here propose a differentiable clustering method that not only will cluster categories in potential clusters/classes, but will simultaneously be building the *feature embedding* while clustering. This is done by optimizing for two main objectives: (1) discriminate an object vs. the background (using the the object proposals information from Section 5.2.1) and (2) maximally separate objects into classes.

Our unsupervised deep clustering approach is inspired by [129]. Their approach is valuable as employing differentiable constraints helps learning in an integrated end-to-end fashion. Our idea is to store the potential cluster means as *weights* in the network and thus have them be *learned*. The proposed clustering will be learned simultaneously with the feature embedding for the input data (Figure 5.3). Namely, given information of a potential object vs. background, captured by a discriminative loss, we further incorporate a differentiable clustering loss (Figure 5.3). Note that this is different from [129], as in our case both our weights and our inputs are changing.

Embedding with clustering: We describe here how to add a clustering-like loss that is integrated in the learning process. To begin with, we conventionally train a convolutional neural network (CNN) to predict foreground and background using objects and background patches, e.g. by using an architecture such as Inception [68] or Network in Network (NiN) [130] (see the experimental sections for details). More specifically, after a number of convolutional layers, we add a differentiable clustering module as shown in Figure 5.3. More specifically, we attach two fully-connected layers, each of 2048 filters. The first fully-connected layer represents the feature embedding x being build and is attached to a classification loss of foreground vs. background. It is also connected to the second fully connected layer where the clustering will be done. Note that we only cluster on the

foreground labels as we do not need to cluster the background activations. The objective function for clustering is shown in Equation 5.1.

$$L_k = \frac{1}{2N} \sum_{n=1}^{N} \text{min}_k[(x_n - w_k)^2] \tag{5.1}$$

In this equation, N is the number of samples, k is the number of defined clusters, w_k is the "weight" (theoretically and typically the mean of the cluster) for each k, and x is the activations from the fully connected layer before the classification fully connected layer. This is differentiable and the gradient descent algorithm is shown in Equation 5.2.

$$\delta w_k = w'_k - w_k = \sum_{n=1}^{N} \begin{cases} l_r(x_n - w_k) & \text{if } k = s(x_n, w) \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

where $s(x_n, w) = \text{argmin}_k[x_n]$ and l_r is the learning rate.

We also add L_2 regularization over the weights to the loss $L_2 = \sum_j w_j^2$. Furthermore, we use a custom clustering metric M_C that informs us if the clusters are evenly distributed as defined in Equation 5.3 and Equation 5.4. This is important since we have no labels to evaluate our data during training.

$$M_C = \frac{1}{N^k} \sum_{k=0}^{K} \sum_{j=k}^{K} |\text{count}_k - \text{count}_j| \tag{5.3}$$

$$\text{count}_k = \sum_{n=0}^{N} \begin{cases} 0 & \text{if argmin}_k[x_n] = 0 \\ 1 & \text{if argmin}_k[x_n] = 1 \end{cases} \tag{5.4}$$

The final loss to be optimized is shown in (Equation 5.5):

$$L = L_k + \alpha_r L_2 + \alpha_c L_C \tag{5.5}$$

where α_r and α_c are hyperparameters which are tuned during the training. L_C is the standard cross-entropy loss for our foreground vs. background classification. For our method, we use $\alpha_r = 0.25$ and $\alpha_c = 0.5$. We apply this loss to every patch labeled as potential object and ignore the background ones. The foreground vs. background task, solved with a standard cross-entropy loss, typically classifies between these two with high accuracy ($>$95%).

During learning, optimization was performed with a 'RMSProp' optimizer, with a learning rate of 0.045, momentum 0.9, decay factor 0.9, and ϵ of 1.0.

A popular alternative algorithm for building embeddings is the contrastive loss [131]. The general loss applied there differs from ours, but more crucially, unlike contrastive loss [131], we only have one embedding space and both losses apply to it without requiring contrastive information.

5.3 Experimental evaluation

We first evaluate our approach in a test where a set of candidate foreground vs. background object proposals are given (Section 5.3.1). This is done because we can evaluate against ground truth proposals and labels, which is not possible in the fully unsupervised case. We then evaluate the quality of the unsupervised proposal mechanism (Section 5.5.1, first part). Finally, we present results from the full method where both proposals and clustering are unsupervised (Section 5.5.1). We here experiment with the challenging Cityscapes dataset [17] described in Section 2.2.

Here, we used the training set for training and the validation set for testing and visualizing results (as the test set has no annotation results). Manual annotation is provided for segmenting classes that are part of the scene such as road, vegetation, building, sidewalk, etc, as well as classes which represent moving agents in the scene, such as pedestrian, car, motorcycle, bicycle, bus, truck, rider, and train. Figure 5.4 shows the sample distribution for these instances. In this work we intend to discover semantic groups from among the

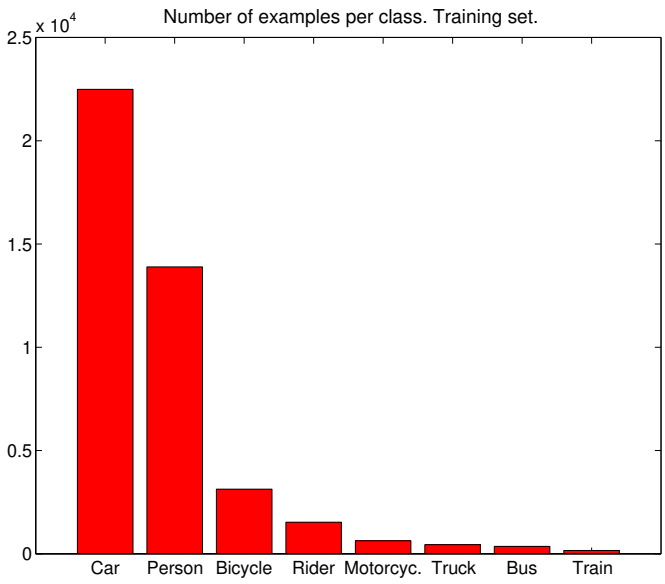


Figure 5.4: Number of examples per class. Cityscapes dataset.

moving objects in these scenes. As seen in the figure, two of the classes, car and person, are the most common ones and significantly outnumber all the other classes.

5.3.1 Unsupervised Memory-based clustering evaluation

We first evaluate clustering, in the case when the ground truth proposal boxes were considered known, but no class labels are given. We wanted to see if important class categories, such as cars, pedestrians, and bicycles can be clustered into semantically meaningful classes. For this and subsequent experiments, the neural network architecture is based on Inception V3 [132]. Since the number of available examples is small (Fig. 5.4), we have pre-trained the network's convolutional layers only as is customarily done in computer vision research. Note that, we *do not* pre-train any of the the fully-connected layers that learn our embedding or clustering. We used the Imagenet pre-training of the convolutional-only layers for simplicity only, but in practice other purely unsupervised features can be used, which have been shown to be of equal or better power [45, 47].

5.4 CIFAR dataset experiments

We first test the proposed unsupervised clustering approach on the CIFAR10 [67] dataset. The goal of this experiment is to test if clustering can uncover separate categories in a simple toy problem with a two class setting.

We selected as an example the dog and automobile classes to label as foreground. We then train a network from scratch based on the Network in Network architecture (NiN) of Lin et al.[130] from scratch for our experiments. All other classes of CIFAR are considered background for this experiment. By attaching our modified clustering objective function to the next to last layer, we wanted to see if the classes are clustered appropriately.

Table 5.1: Contrastive loss on two embeddings on CIFAR10 for discovery of two classes. Per cluster accuracy (in %) for each of the two given classes on the test set (class labels are not used during the training process).

Clusters	Automobile	Dog
Cluster 0	0%	100%
Cluster 1	0%	100%

We can see in our simple experimental results that classes are naturally clustered with the majority of examples correctly assigned. Table 5.2 shows quantitative results on the test set are shown. As seen around 70% (68.5%) of the automobile classes are correctly clustered, and 80% (82.1%) of the dog examples are correctly assigned to a cluster. Note that in these cases, the concepts and classes of dog and automobile are unknown to the training algorithm and we are just looking at them post clustering to see if any natural clustering is occurring. The method naturally splits the classes into two different means despite not being given labels.

Figure 5.5 further visualizes the training process. It shows classification accuracy per each newly learned cluster for each of the given classes. This is assuming the ground truth labels are known and is used for visualization purposes only; the labels are not used in training. We can see that after a number of iterations, the algorithm starts to correctly

assign the majority of dog examples (Class 1 in CIFAR) to cluster with an id 0 and the majority of automobile examples (Class 5 in CIFAR) to cluster number 1.

We also experimented with the *contrastive loss* [131] in this setting but it did not yield good clustering results (all examples are clustered in a single class). We saw this behavior consistently when attempting to cluster using contrastive loss as seen in Table 5.1.

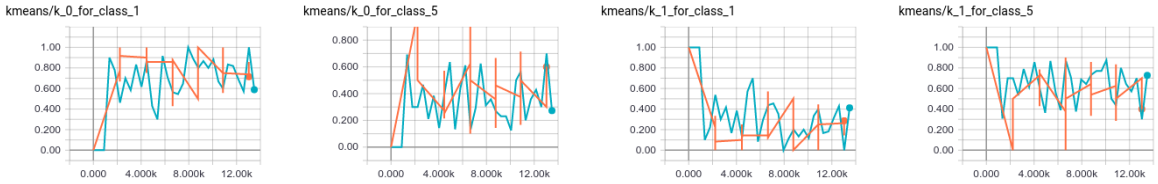


Figure 5.5: The clustering of the dog class (Class 1 in CIFAR) and automobile class (Class 5 in CIFAR) while training on the CIFAR10 dataset. The full training process is shown until about 12,000 steps. As seen, the newly formed cluster 0 (first two figures to the left), tends to attract much of the dogs examples, Class 1 in CIFAR, (first figure), than automobile, or Class 5, examples (second figure). At the same time, the newly learned cluster 1, does not classify correctly dog examples (third figure), but does well on automobiles (fourth figure). Blue/Orange curves denote training vs. test data. The training has no notion of labels, they are used here only for evaluation purposes.

Table 5.2: Unsupervised clustering results on CIFAR10 for discovery of two classes. Per cluster accuracy (in %) for each of the two given classes on the test set (class labels are not used during the training process).

Clusters	Automobile	Dog
Cluster 0	68.5%	17.9%
Cluster 1	31.5%	82.1%

5.5 Cityscapes experiments

Next we show experiments on Cityscapes with clustering of the eight categories mentioned above, pedestrian, car, motorcycle, bicycle, bus, truck, rider, and train, into a small number of clusters (e.g. 3 since most categories are rather rare in the data Figure 5.4). We report the results in terms of the number of object patches classified in each cluster. We here test two methods: 1) the baseline clustering approach and 2) our Unsupervised Memory-based

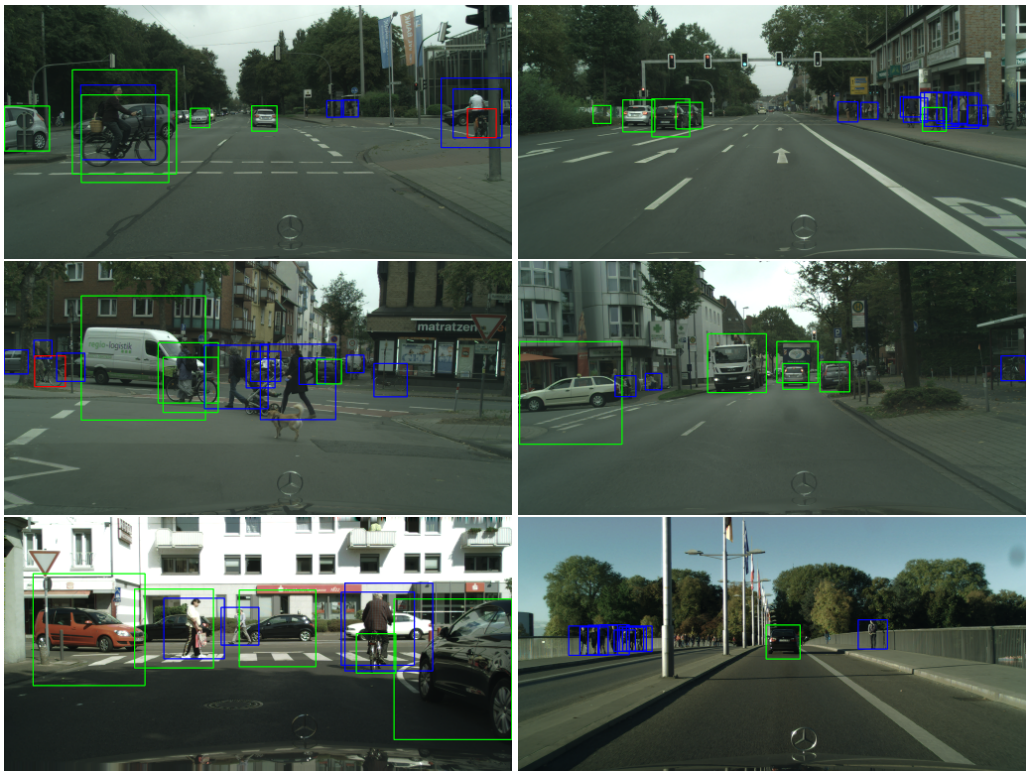


Figure 5.6: Visualization of clusters learned by UMC when proposals are given (for K=3). It is clear that the green class is responsible for retrieving cars, the blue one persons, and the red one bicycles. We can see that both cars and persons are discovered pretty well. Bicycle is much more rare class and can be confused with a person or with a partially visible car in the background. (Best viewed in color.)

Clustering (UMC). In order to put them on equal footing, they both have the same pre-trained features of the early convolutional-only layers. The features for the fully-connected layers are not pre-trained and are initialized randomly. Both methods have the same feature capacity (and also input data), and the only difference is that the UMC is applying the proposed memory-based clustering loss, proposed in Equation 5.5.

Clustering of all examples in 3 clusters for both settings are shown in Table 5.3 (see Appendix for the baseline). This table is not normalized to be more informative. As seen, the proposed embedding forms clusters where one tends to cluster the majority of examples from person and rider, and the second one clusters more vehicle-like objects car, bus, and motorcycle. Classes such as bicycle and train are more uncertain. In the case of three classes, a bicycle cluster forms but is a weaker one, and while it still attracts a number of

bicycle examples, it is split between the bigger classes. This can be attributed to the fact that the bicycles class is very small (its examples represent about 1/10 of cars and 1/7 of persons); also bicycles and persons often appear as a foreground to a car-like object. We can also observe that categories that are related such as person (aka pedestrian) and rider (defined as a person riding a motorcycle) are assigned to the same cluster, whereas car, motorcycle, truck, and bus are assigned to another one. We also note that while motorcycles may seemingly be more related to bicycles, appearance-wise they share more visual features with cars, e.g. tires, fenders, etc.

The baseline embedding trained on the same data without our clustering method does not yield as good clustering results (see Table 5.4 for a summary). We saw this behavior consistently. We note that it obtains the same input information but does not use additional clustering objective. We also experimented with the contrastive loss [131], which is a popular alternative, but was not successful and yielded one-sided clustering results as shown in Table 5.1. Table 5.4 summarizes the performance for both embeddings, by computing the clustering purity index. In this experiment can see that our embedding is showing better performance on this 3-class classification task. We further observed that an improved clustering initialization based on the idea of kmeans++ (denoted as Cl++ here) improves the clustering accuracy. We however do not use that in subsequent experiments as the initialization itself is time consuming.

Figure 5.6 visualizes the three retrieved clusters (color-coded) when clustering with our approach, We can see that people (in blue) and cars (in green) are often correctly retrieved. Bikes are more rare and may be more often mistaken, for example in cases where a portion of the patch contains part of a car, or since the bicycle very often has a person riding it. In these visualizations we also observe that the proposed embedding provides adequate representation for clustering. This is a promising result, given that it is learned by not providing a single class label during training.

Table 5.3: Unsupervised clustering of objects from the Cityscapes dataset with 3 clusters with our clustering loss. The table shows number of examples assigned to each learned cluster (K=3). The clustering purity index is 61.99% for K=3 and the mean accuracy 76.16%. The baseline index is 33.27% and 50.61%, respectively.

Classes	**Cluster 0**	**Cluster 1**	**Cluster 2**
Person	**4302**	198	29
Rider	**634**	161	17
Car	690	**5053**	538
Truck	34	92	10
Bus	25	117	7
Train	12	23	3
Motorcycle	73	119	21
Bicycle	583	946	180

5.5.1 Evaluation of the full unsupervised method

In this section we examine what kind of proposals are retrieved from the full self-supervised method and what classes are being formed. This proposal mechanism requires no manual supervision.

Object proposals generation evaluation: We first evaluate the object proposal generation mechanism. Figure 5.7 shows example object proposals. The generated proposals constitute regions of objects instances and very often contain an actual object that is present in the scene. As noted, no manual supervision has been used in this approach and no objectness, other external labels, or features are being used at this stage of the algorithm.

Table 5.5 shows recall of our algorithm with respect to *known* ground truth mask proposals from Cityscapes (with IOU on segments of 0.5). While the retrieved object pro-

Table 5.4: Unsupervised clustering results on Cityscapes dataset. Clustering purity index for K=3 on the validation set (class labels are used in evaluation only).

UMC embedding (ours)	UMC embedding (ours, w. Cl++)	Baseline embedding
61.99%	69.98%	33.27%

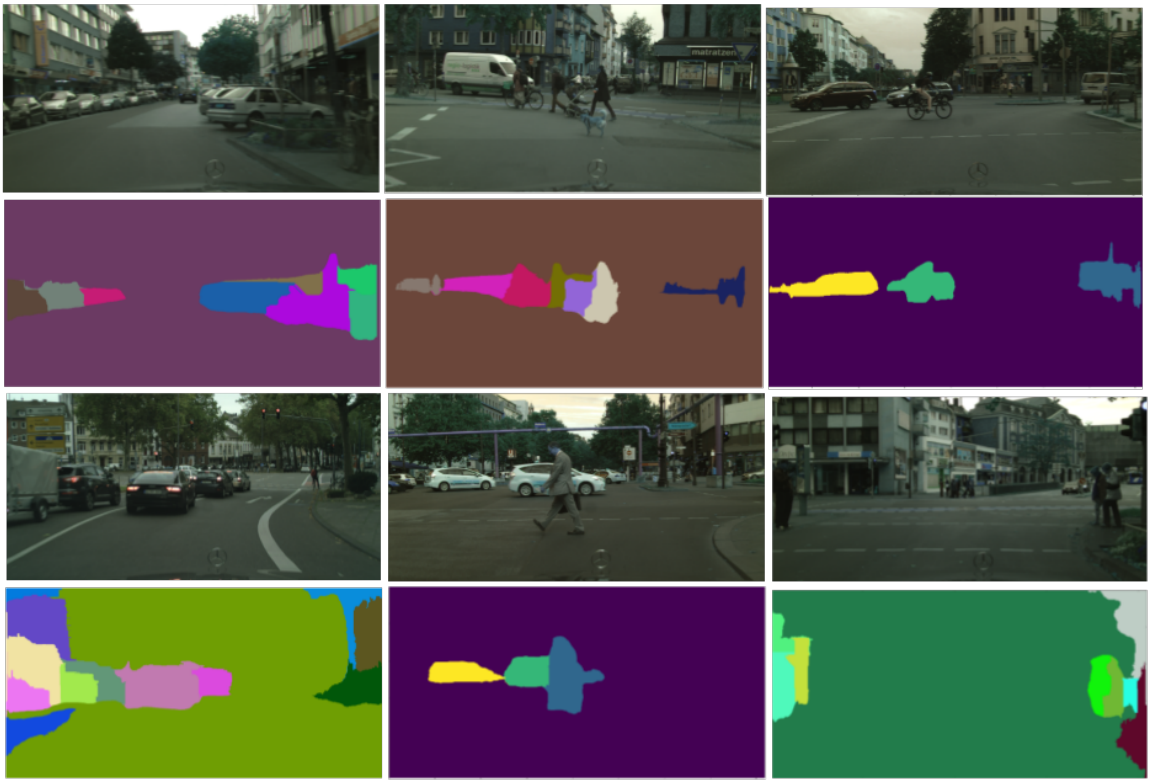


Figure 5.7: Example proposal instances generated from unsupervised depth and clustering in RGBD space.

posals are meaningful, the recall numbers with respect to ground truth objects in city environments such as person, bicycle, car, are low, with 8-20 percent recall of all available objects, especially compared to powerful detectors, e.g. [71]. This is not surprising given the small number of average proposals per image (5.4). Still it is interesting that the proposals have recalled many of the common objects available, and serves our main purpose (in next section) in which we learn to group semantically related objects in classes without labels. Another interesting observation is that the proposals can also separate individual objects (i.e. instances) without any pre-training, knowledge of objectness, or trained detection. Figure 5.7 shows examples of generated proposals where the generated proposals are backprojected to the image.

UMC Clustering on automatically generated proposals: We now test the final clustering, when proposals are unsupervised, i.e. automatically generated. We here test the

Table 5.5: Recall of our proposal generation algorithm on Cityscapes objects (in %). Bottom row shows all examples total available in the data.

Person	**Rider**	**Car**	**Truck**	**Bus**	**MotorCyc.**	**Bicycle**	**Vegetation**	**Building**	**Train**
8.9	15.8	21.2	17.2	20.4	10.7	8.2	2.8	4.6	0.0
3458	544	4765	93	98	149	1285	486	491	23

baseline algorithm against the following algorithms.

1. Baseline algorithm, unsupervised proposals. Table 5.7

2. Unsupervised Memory-based clustering, unsupervised proposals using our proposed algorithm. Table 5.6

3. Features pre-trained with supervision, unsupervised proposals. Table 5.8

The latter algorithm is directly using features which have already been pre-trained on Imagenet. They are applied to each of the bounding box proposals. These features are supervised as they have been previously trained to separate classes on Imagenet.

Table 5.6: Our clustering on Cityscapes validation set with unsupervised proposals.

Classes	**Cluster 0**	**Cluster 1**	**Cluster 2**
Person	3	47	**100**
Rider	6	5	7
Car	**471**	273	23
Truck	13	10	1
Bus	16	17	5
Train	2	5	0
Motorcycle	0	3	2
Bicycle	11	23	4
Vegetation	187	134	39
Building	333	404	174

Table 5.6 shows the results of our unsupervised clustering with unsupervised proposals. Since the proposals and clusters are all unsupervised, we compare for evaluation purposes

with respect to the ground truth on the Cityscapes validation data. This is a form of out-of-sample testing, since our unsupervised proposals do not match the ground truth objects. That is, we are comparing only against the correctly recalled objects but there may be others that are correctly clustered (as we see later) whose bounding box is somewhat mismatched. We can see that meaningful clusters have formed, e.g. one class has attracted more cars, another most of the persons. Figure 5.8 visualizes the results of this clustering. Here the clustering results from Table 5.6 are more clear now, as we observe that, although noisy, the algorithm has formed interesting semantic clusters. For example, clearly the cluster marked in red, although sometimes suffering from a poor proposal box, has definitely learned a person detector. Similarly there is a cluster focusing on cars (blue), and interestingly it contains mostly side or back cars, whereas the green cluster (which has attracted many of the recalled building patches) also contains frontal car examples.

Table 5.7: Baseline clustering on Cityscapes validation, unsupervised proposals without our loss.

Classes	**Cluster 0**	**Cluster 1**	**Cluster 2**
Person	138	7	5
Rider	15	2	1
Car	690	39	38
Truck	21	2	1
Bus	35	1	2
Train	5	1	1
Motorcycle	5	0	0
Bicycle	34	0	4
Vegetation	323	18	9
Building	789	47	75

Table 5.7 shows the clustering of the counterpart baseline method trained on our proposals without our clustering loss where we see that clustering is more one-sided and of poorer quality (see later Table 5.9 for comparison metrics).

Table 5.8 shows the corresponding clustering with *supervised features*. It can be seen as an upper bound on clustering when the embedding space was being supervised by labels.

Table 5.8: Clustering on Cityscapes validation, unsupervised proposals with supervised features

Classes	**Cluster 0**	**Cluster 1**	**Cluster 2**
Person	0	16	**134**
Rider	0	5	13
Car	**734**	22	11
Truck	4	20	0
Bus	11	24	3
Train	1	6	0
Motorcycle	0	2	3
Bicycle	2	8	**28**
Vegetation	44	**297**	19
Building	131	**513**	267

Here it is interesting to observe that thanks to the label supervision this method clusters better and puts together patches of pedestrian, rider, bicycle, which contain a person.

Table 5.9: Clustering results with unsupervised proposals. Clustering Purity Index. Our unsupervised method outperforms the baseline method which uses the same inputs. We compare to clustering with supervised features as an upper bound.

Clusters	(car,person,bicycle)	(car,person,building)	(car,person,vegetation)
Supervised feat.	0.92	0.80	0.87
Baseline unsup.	0.80	0.47	0.61
Ours unsupervised	**0.87**	**0.56**	**0.67**

Table 5.9 summarizes the results of the above tables reporting the clustering purity index for several groups of classes. As seen our method outperforms the baseline in this setting too. Clearly adding supervision is helpful to partition an embedding well. However, it is interesting to note that this clustering loss enables us to recover some of that partitioning without any supervision.

While the results are of course noisy, we can see that our algorithm has actually managed to learn about the existence of semantic categories and has grouped them into separate classes, e.g. persons, cars. This is an exciting result as it is done without any labeling. We

believe these results are important as they illustrate what can be learned without any manual supervision. This shows the effect of this clustering loss on embeddings learned with deep methods

5.6 Conclusion

In this chapter, we explored the challenging task of self-supervised object category learning. We discover semantically meaningfully classes in a challenging dataset, without manual supervision. Specifically, we propose a novel proposal generation that takes the clustering into the 3D RGB space, and a differentiable clustering objective which learns to separate classes during learning and builds a better embedding. This validates our Thesis Statement (1.1) as the 3D features are key to this work being possible. The clustering idea is to learn the clusters which are stored as weights, and simultaneously learn the feature representation while clustering the data. We showed that with this approach semantically related objects can be retrieved and 'detected' without supervision. This is the first result of its kind demonstrated in a challenging setting, the Cityscapes dataset.

In the next chapter, we will instead start exploring supervised methods to leverage this type of RGBD data instead of the unsupervised methods in the last two chapters.

Figure 5.8: Main algorithm results: Visualization of clusters learned by the unsupervised procedure when the proposal boxes are also unsupervised. Several interesting clusters are formed: person is retrieved and clustered in the red cluster; side and back view of a car tends to be in the blue cluster, and the green cluster contains buildings,other structures, and also the front view of cars. Some noisy classifications and proposals are also present (e.g. right row, bottom two). Best viewed in color.

CHAPTER 6
SEMANTIC INSTANCE LABELING LEVERAGING HIERARCHICAL SEGMENTATION

Up until now, we have focused on unsupervised and self-supervised techniques. However, they have several limitations as pointed out in Chapter 5 and even though those techniques were impressive, we were never able to match supervised accuracy. In this chapter, we discuss using depth, xyz position, surface normals, and color to semantically label distinct regions in a 3D point cloud. We slightly modify the method from Chapter 4 to work with surface normals instead of depth and we show that to be a better shape cue for this type of semantic labeling as proposed in our Thesis Statement 1.1. This method is also near real-time with results shown on a robot in the video [133].

6.1 Introduction

Much work has been done on dense semantic labeling and indoor scene understanding for the use of robots. However, these are often limited to scene recognition, limited classes, or structural information. With the introduction of affordable RGBD cameras such as the Microsoft Kinect, dense point clouds can be constructed in indoor environments with minimal effort. This kind of data has changed the way we do object detection and labeling. With this type of data, we can use both shape and color as cues for semantic labeling in order to improve conventional methods as mentioned in the Thesis Statement (1.1). Although object recognition has come a long way with these new sensors, indoor semantic labeling and scene understanding is still in it's infancy. For robots to effectively interact with an indoor environment autonomously, they need access to accurate semantic information about their environment. Some work has been done with labeling of SLAM maps [76, 73, 134, 74, 75]. However, most previous work doesn't focus on dense semantic maps to allow a robot

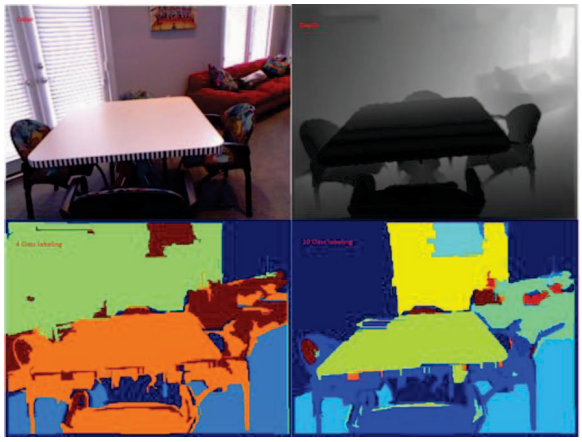


Figure 6.1: Semantic segmentation and labeling of 3D Point clouds. Top: RGB and depth maps used as input in the algorithm. Bottom: The 4 Class output and 10 Class output as described in Section6.3

to interact with the environment.

With the introduction of the NYU Dataset[15], we now have an indoor densely labeled Kinect dataset for scene understanding. Our implementation aims to tackle the problem of accurate, dense, and fast semantic segmentation. Our method uses hierarchical segmentation to construct meaningful features from full objects and not patches or individual pixels. The assumption being that whole objects will have more meaningful features (shape, size, color) than superpixels, patches, or pixels that correspond to parts of objects. We evaluate our method using the NYU Dataset[15] and segment each frame as either floor, structure, furniture, or props. We additionally segment each frame using more specific classes to label objects such as bed, chair, etc. based on [78]. Our method is novel in that we focus on instance labeling based on a hierarchy of segmented super pixels. This has many advantages to previous work using bounding boxes or pixel-based detectors.

We aim to not only label each segment as belonging to a category but to separate each segment as different objects even if they belong to the same category. In a scene, we can not only know what pixels are labeled as furniture, we can know how many pieces of furniture there are, where they are, and what pixels belong to each one. Because of this, we avoid the pitfalls of bounding box detectors in that we know the contour of the object and we

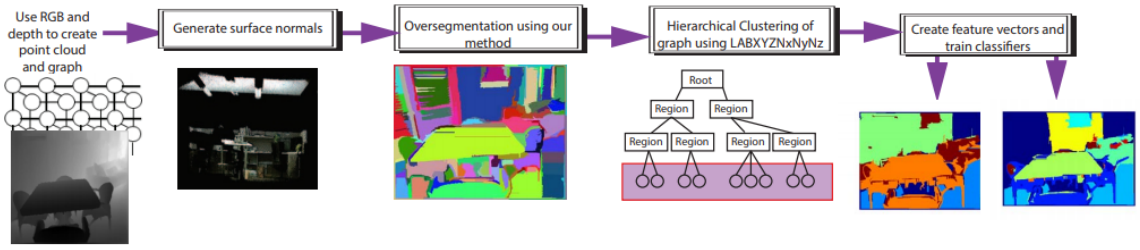


Figure 6.2: An overall schematic of our full method. First we generate the point cloud and a graph, then we generate surface normals, then we over segment the point cloud, create a hierarchy of super pixels, and finally train our feature vectors using a random forest of decision trees.

avoid the pitfalls of general semantic segmentation since we generate instances of objects. Another advantage arises due to this; since we test on larger, merged segments from super pixels, our testing time can be very fast.

6.2 Method

Our method was the first to use a hierarchy of super pixels to train a classifier on the NYU V2 dataset. We rely more on the shape, color, surface, and position of objects than hand-crafted features like SIFT or pixel-based metrics like conditional random fields or convolutional neural networks. The theory behind this is that people learn objects based on these characteristics and the shape and color cues can help each other like discussed in our Thesis Statement. This methodology requires a higher level segmentation than just super pixels. We use a variant of the hierarchical segmentation method based on our previous work[1] in Chapter 4 in order to create segments that match objects instead of patches. The full method is shown in Figure 6.2.

6.2.1 Fast Surface Normal Estimation

As Kinect data is extremely dense, we use surface normals to help segment the 3D data and train our classifier. We use the surface normal estimation described in [128]. It uses integral images to estimate dense surface normals at every point for projective devices and is imple-

mented in PCL [135]. Surface normals allow us to enhance the segmentation of Chapter 4 and still maintain the non-linear combination of color and depth. Our enhancement of the method is described in Section 6.2.2.

6.2.2 3D/4D Segmentation

This segmentation method is based on our method described in Chapter 4 that uses a hierarchical graph-based segmentation based on [18]. Although there are other methods that segment 3D data such as [25, 32, 29, 33, 26]. We modify our previous method to allow a single, non-linear edge weight. This new approach differs from Chapter 4 in that we construct a graph using only 8-neighbors (ignoring time and using only voxels) whilst maintaining non-linearity. We use the surface normals estimated in Section 6.2.1 and the color to create one edge weight W:

$$W = \max\left(\sqrt{N(x,y,z)^2 - N(x',y',z')^2}, \sqrt{C(x,y,z)^2 - C(x',y',z')^2}\right) \tag{6.1}$$

where $N(x,y,z)$ is a function that yields the surface normals from the point cloud, $C(x,y,z)$ is a function that yields the colors from the point cloud and $(x,y,z) \in N(x,y,z)$, where $N(x,y,z)$ is the neighborhood of voxel (x,y,z). We then over-segment the same way as in Chapter 4 in Section 4.2.1.

6.2.3 Hierarchical Construction

After the over-segmentation described in Section 6.2.1, we compute feature vectors for each super-voxel that we use for hierarchical propagation and for training our semantic classifier. For feature vectors we extend the feature space of Chapter 4 by using histograms of CIE Lab color, histograms of 3D position, and histograms of 3D surface normals, called LABXYZNxNyNz histograms, using all the voxels in each region. As in Chapter 4, rather than computing large multi-dimensional histograms, we compute 1- dimensional

histograms for each feature. It is important to note that we have added surface normals as features here and are ignoring optical flow as we are just looking at single RGBD frames. We use 20 bins of LAB color features and 30 bins of the XYZ position features just like in Chapter 4, adding 30 bins of the NxNyNz surface normal features. We also compute the region's 3D size, height, width, 2D centroid, 3D centroid, 3D minimum, and 3D maximum as extra features. Using the aforementioned feature vectors, the S-graph (from Section 4.2.2) is created to propagate the hierarchy. The graph is comprised of vertices that are each super-voxel region and edges that are formed to join neighboring regions using Equation 6.2, which has been modified to use surface normals instead of optical flow as we don't have video here and shape is an important feature. We then construct a dendrogram that is a hierarchical tree of super-voxel regions defined by similarity where the root node is the entire-set merged and the leafs are the supervoxels computed by Section 6.2.2. To match the regions we use a modified version of the SAD difference equation in Equation 4.6 that adds surface normals instead of optical flow. The difference between Region R and Region S is defined as:

$$\Delta H = \sum_{i=1}^{\text{NUMBINS}} \sum_{\beta \in (l,a,b,x,y,z,N_x,N_y,N_z)} |\frac{R_\beta[i]}{R_N} - \frac{S_\beta[i]}{S_N}| \tag{6.2}$$

where R_N is the number of voxels in Region R and S_N is the number of voxels in Region S. We use the same parameters defined in Chapter 4 with a lower tree cut of 0.15.

6.2.4 Feature Selection

Feature selection for this task is extremely complex and it is difficult to determine which properties might be important. The objects in each category can vary quite a bit and it is difficult to tell what features are best for this classification. Some semantic segmentation methods [15, 74, 31] also use custom, expert features to label classes using the NYU dataset [15] while others use machine learning techniques to learn features such as Couprie

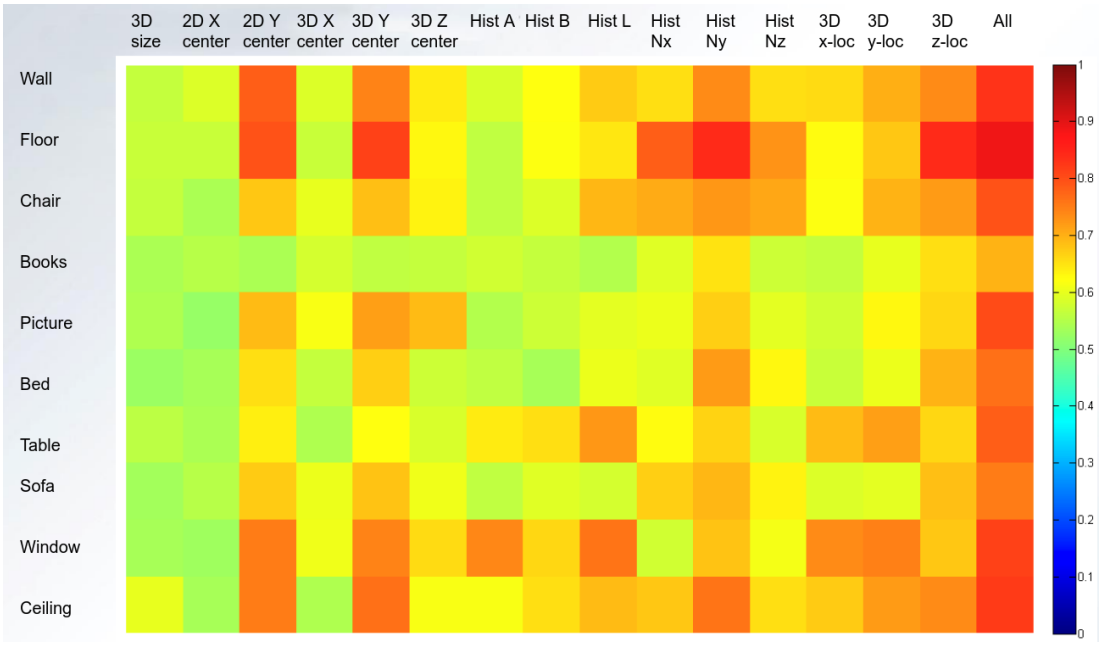


Figure 6.3: Classification Accuracy for different 3D Features on a validation set as described in Section 6.3. Heat mapped with blue being 0% and red being 100%

et al.[78], which uses a convolutional neural network to learn important features. The arguments between the benefits and limitations of these approaches are beyond the scope of this work. For each region, we use general features such as shape, size, position, orientation, surface normals, and color in order to catch any important features. In addition we test the features described in Section 6.2.3 as well as the impact of adding a 1000 cluster bag-of-words of SIFT feature points. A random forest of 500 decision trees yielded the best results on our test using the NYU Depth Dataset V2 [15]. When testing all the different features, a random forest was the most accurate classifier. This makes some amount of sense given that we are using histograms and the features are very independent of each other. This was the reasoning behind using a random forest as opposed to an SVM or neural network.

For the histogram features described in Section 6.2.3, we ran several classifiers on each different feature as a binary classifier using a random validation set from the training data of the NYUDv2 dataset[15]. The results are shown in Figure 4 with 1 being 100% accuracy and 0 being 0% accuracy for the top 10 classes where each column is a feature from a subset of our features (size, 2D X centroid, 2D Y centroid, 3D X centroid, 3D Y centroid,

3D Z centroid, A histogram, B histogram, L histogram, Nx histogram, Ny Histogram, Nz Histogram, 3D X histogram, 3D Y histogram, 3D Z histogram, and all combined features) and each row is a class. The last column is a combination of all the features and is shown to do the best. Not only did we find that the combination of all features yields the best result, we also found that random forests[136] always yield the best accuracies for our data. This confirms our hypothesis that random forests will work well in this scenario.

In addition, we found that the histograms of normals are more representative features for semantic labeling than depth. This can be seen in Figure 6.3 comparing the Hist Nx, Ny, Hz columns with the 3D Z center column or the 3D z-loc column. Note that the histogram of normals is almost always achieving higher classification accuracy than the depth (Z columns). This supports a part of our Thesis Statement which proposes that semantic label classification "**improves with surface normals more than with depth**."

6.3 Experiments

We tested our method on two different tasks. The first is semantically labeling everything as either floor, structure, furniture, or props as proposed in [15]. This is shown in Section 4.1 and reveals how our method generalizes categories accurately compared to state of the art. The second test is labeling the 13 most frequent classes, with the 14th being other, in the NYU V2 dataset as proposed by [78]. This is shown in Section 6.3.2 and reveals how our method does on more specific objects and classifications.

6.3.1 4 Class Test

After determining our features and classifier, we ran our method on the NYU V2 dataset [15] and trained the 4 Class and 14 Class classifiers to compare against previous implementations .We further experiment by training our method on the 4 Class set with and without SIFT [137]. Previous methods such as [77, 15, 74] use a bag of words (BOW) of SIFT features[138] to help classify. Being uncertain about the usefulness of SIFT in such a gen-

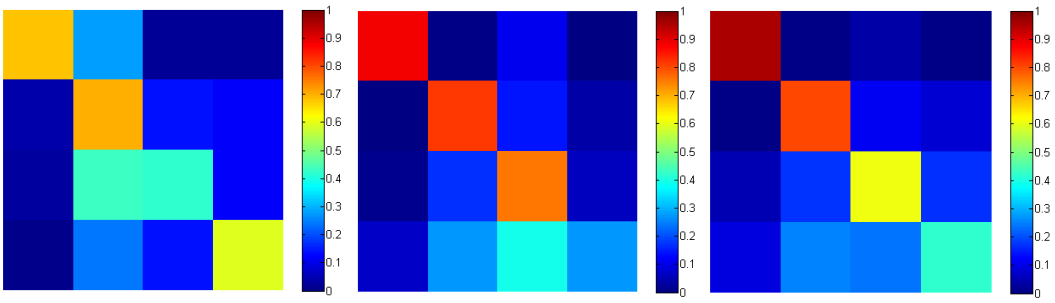


Figure 6.4: Confusion Matrix for 4 classes of NYU Dataset Left: [15], Center: our method with SIFT, Right: our method without SIFT. Heat mapped with blue being 0% and red being 100%

eral classification method, we trained our method with and without it as a test. The class pixel accuracy confidence matrices for our method (with and without a BOW of SIFT) and the method from [15] are shown in Figure 5. It is around 10% more accurate although we suffer categorizing props/objects, showing no improvement over the 42% of [15]. In table 1, our method is compared to other state of the art methods showing us to have better accuracy in ground, furniture, overall class accuracy, and overall pixel accuracy. Note that we do not compare to [72] in this table since they use a different accuracy metric then all of the other methods testing on this dataset.

6.3.2 14 Class Test

We also test our method using the 14 classes (including other) proposed by Couprie et al.[78]. It is important to note that this classifier was trained independently of the 4 class classifier does not use the classifications shown in Section 4.1 although that may improve classification results. For this test, we did use a BOW of SIFT features as we are finding specific objects. We compare here against Wang et al.[64] and Couprie et al.[78], which are both unsupervised learning methods as opposed to out method which uses hand-crafted features. As shown in Table 6.2, we improve total class accuracy by more than 5% over Wang et al.[64] and over 11% more than Couprie et al[78]. We do slightly worse in a couple categories such as bed, sofa, wall, books, and TV. It is uncertain whether our performance

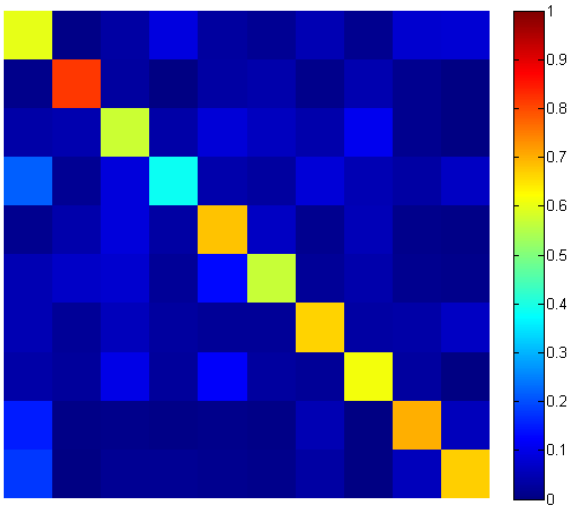


Figure 6.5: Confusion Matrix for the top 13 classes of NYU Dataset[15] as proposed by [78] (ignoring the 14th class other). Heat mapped with blue being 0% and red being 100%

increases are due to the hand-crafted features or the hierarchical segmentation. However, since Silberman et al.[15] use a much larger set of features that include some of ours, and both methods outperform others, we suspect that most of our accuracy gain comes from using our hierarchical segmentation.

6.3.3 Results

To compliment the quantitative results shown in Table 6.1 and Table 6.2, we show some qualitative results of our algorithm in Figures 6.7 and 6.6. Figure 6.7 shows the result of our algorithm on the 4 general classes and Figure 6.6 shows the result of our algorithm on the 14 specific classes. For both tests, we excel at labeling different structures, the floor, the ceiling, and different furniture. However, our algorithm suffers labeling objects. This could be due to the hierarchical segmentation. It is possible that small objects get merged to larger segments, which would label them as part of the furniture or structure near the object. It is also possible that our generic features involving shape and surface normals do not generalize well for objects. The lack of textural information could also be contributing to the lack of object accuracy. However, as shown in Table 6.2, the hierarchical segmentation does allow the algorithm to excel at recognizing the furniture and the structure. Our method

is far superior at recognizing chairs, the ceiling, the floor, tables, and windows. These all have relatively similar shapes, sizes, and surface normals which can be leveraged as context as we say in our Thesis Statement. Graph-based segmentation methods also tend to match the boundary of more rectangular objects, creating better segments for planar objects, which contributes to our high accuracy in these areas.

Table 6.1: A comparison of the per-pixel and per-class classification accuracy of the 4 class set comparing our algorithm to other state of the art methods.

	Ground	Furniture	Props	Structure	Class Acc.	Pixel Acc.
Silberman et al.[15]	68	70	42	59	59.6	58.6
Couprie et al.[78]	87.3	45.3	35.5	86.1	63.5	64.5
Cadena et al.[139]	87.9	64.1	31	77.8	65.2	66.9
Stuckler et al.[82]	90.7	68.1	19.8	81.4	65.0	68.1
Wang et al.[79]	90.1	46.3	**43.3**	81.4	65.3	N/A
Our Method (No SIFT)	**95.3**	60.9	42	80.2	**69.6**	69.5
Our Method	88.5	**75.5**	27.1	**81.8**	68.2	**71.8**

.

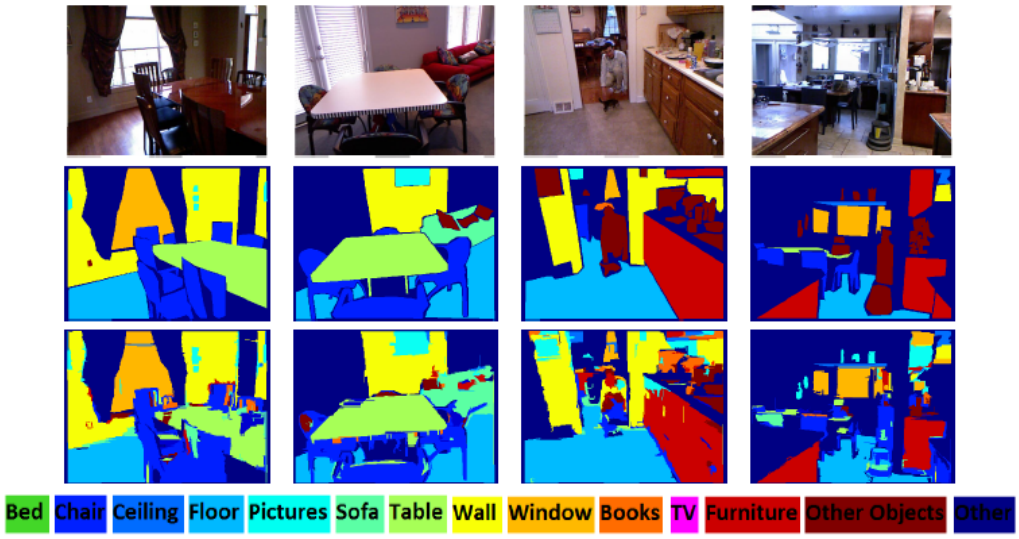


Figure 6.6: Results for the 14 class set proposed by [78] The top row shows RGB, the middle row shows the ground truth labels, and the bottom row shows our classification.

Table 6.2: A comparison of the per-pixel and per-class classification accuracy of the 14 class set proposed by [78] comparing our algorithm to other state of the art methods.

	Couprie et al.[78]	Wang et al.[79]	Our Method
bed	38.1	**47.6**	33.5
chair	34.1	23.5	**53.3**
ceiling	62.6	68.1	**84.8**
floor	87.3	84.1	**92.8**
picture	40.4	26.4	**55.3**
sofa	24.6	**39.1**	36.8
table	10.2	35.4	**40.1**
wall	**86.1**	65.9	75.6
window	15.9	52.2	**59.8**
books	13.7	**45.0**	20.4
TV	6.0	**32.4**	27.3
Furniture	**42.4**	16.7	21.5
Objects	8.7	12.4	**17.6**
Class Acc	36.2	42.2	**47.6**

6.4 Conclusion

In this chapter, we presented a new approach to indoor semantic labeling that uses a modified version of the hierarchical segmentation method from Chapter 4 with general features pertaining to shape, size, and color in order to generate meaningful segments. We then use a random forest of decision trees in order to train a classifier to classify both general categories and more specific categories on the NYU V2 dataset. We show that our results are comparable and even improve on state of the art methods including those that use convolutional neural networks trained only on this small dataset, super pixel methods, and those methods using advanced structural features. This validates the thesis statement that shape can be used to improve object classification. In our work in the following chapters, we will extend this using newer, more advanced convolutional neural networks trained on several datasets. We will also use these datasets for surface normal prediction.

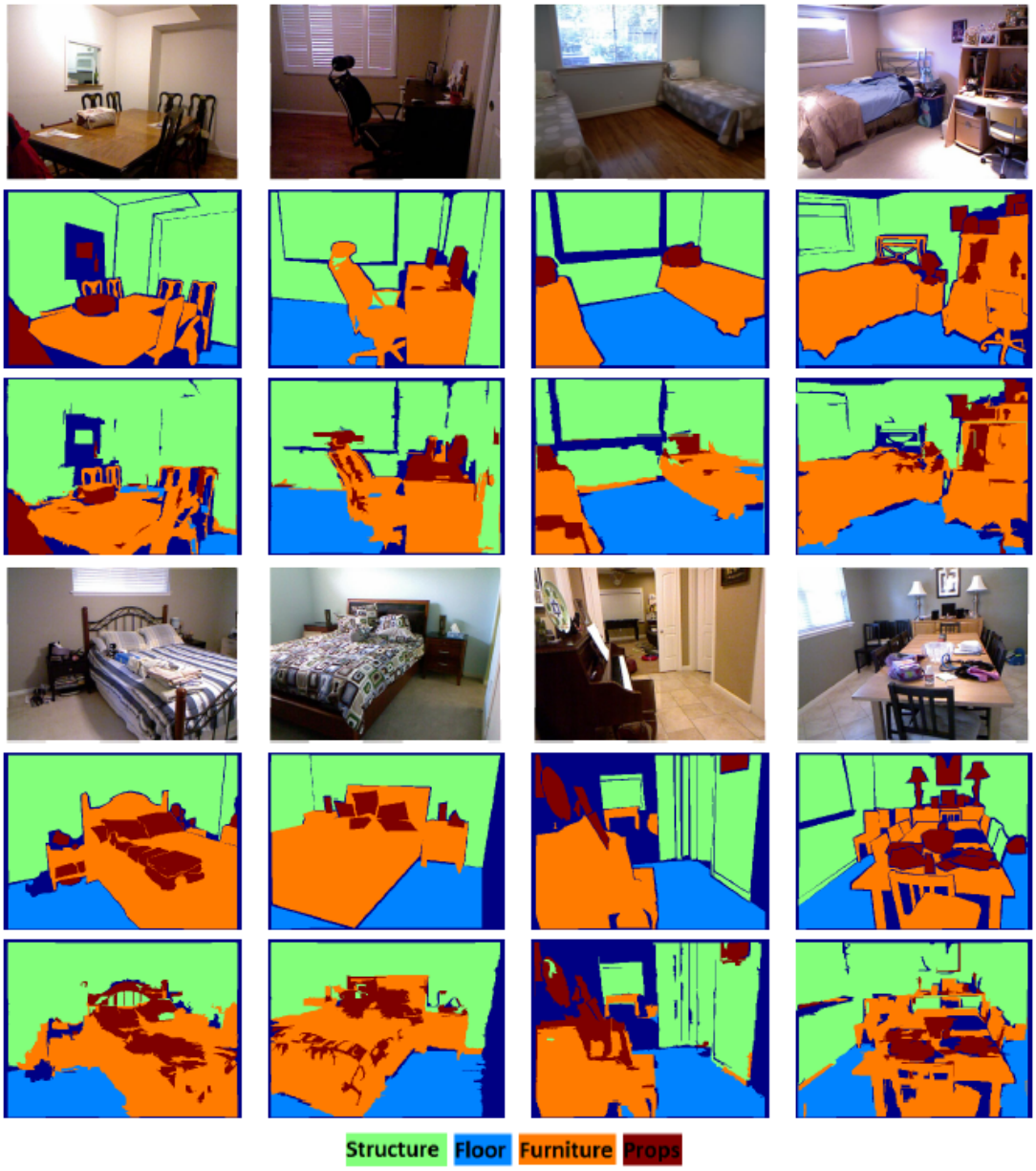


Figure 6.7: Results from the 4 Class set from [15]. The top row shows RGB, the middle row shows the ground truth labels, and the bottom row shows our classification.

CHAPTER 7
LEVERAGING SEMANTICS FOR REAL-TIME SURFACE NORMAL PREDICTIONS

This chapter continues our discussion about semantic labeling of indoor environments using 3D contextual information. In the previous chapter, we used depth as an input and constructed hand-crafted RGBD features from our unsupervised segmentation from Chapter 4. In this chapter, we will instead take only a single RGB image as an input and predict both the shape (as surface normals) and the semantic labels. The previous chapter lacks the effectiveness and ability of deep learning; whereas here we can learned shared-feature representations for both RGB to surface normal and RGB to semantic labeling tasks. Our focus in the previous chapters has been only on clustering or predicting semantic labels using shape as an input. In this chapter, we will instead focus on predicting shape as an output.

7.1 Introduction

Here, we address the problem of learning a model that can predict surface normals and semantic labels for each pixel, given a single monocular RGB image. This has many practical applications, such as in augmented reality and robotics. A video demo of an AR task doing this using our method can be seen here [140].

Most high-performing methods train deep neural networks to perform the task of estimating surface normals using large training sets. However, an often overlooked aspect of such approaches is the quality of the data that is used for training (and testing). We have found that the standard techniques for estimating surface normals from noisy depth data, such as the widely used method of Ladicky et al. [86], can result in inconsistent estimates for the normals of neighboring points (see Figure 7.4 for an example).

We propose a simple technique to fix this, by regularizing the prediction of normals that correspond to the same surface. This encodes our intuition that floors should be flat and pointing up, etc. To estimate which pixels belong to the same surface, we leverage the fact that many depth datasets also have per-pixel semantic labels. This in itself does not tell us which facet of the object a pixel belongs to, but we use simple heuristics to solve this problem, as described in Section 7.2.1. See Figure 7.1 for an illustration of the benefits of this approach. In this chapter, instead of using depth or shape to improve semantic labels, we reverse this and show that semantic labels can fix and improve surface normal prediction. This shows the joint relationship between shape and semantics as mentioned in the Thesis Statement (1.1).

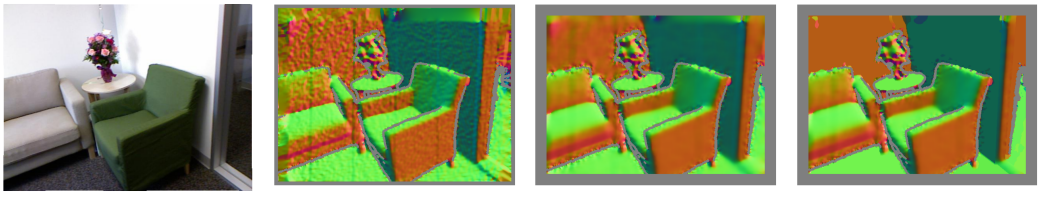


Figure 7.1: Visualization of different ways of computing "ground truth" normals. Top left: a sample image from the NYUDv2 dataset. Top-right: computed using method of [89] that uses a small smoothing window. Bottom-left: results of our method using larger smoothing window. Bottom-right: results of our method after semantic smoothing (if labels are available). Note that the back and right wall are cleaned up to a large degree due to this correction.

Unfortunately, even after such "data cleaning", many real world datasets are still too small to train deep models to their full potential, so it has become popular to leverage synthetically generated images. These are noise-free, but it is not obvious how best to combine real and synthetic data. The standard practice (e.g., [14]) is to pretrain on the synthetic Scenenet dataset and then fine tune on the real dataset. We propose a simple improvement to this idea, which is to train the model on a carefully chosen mix of real and synthetic images in each mini-batch. This simple insight improves results considerably, as we show in Section 7.3.

In addition to improving the way data is used, we propose some improvements to stan-

dard modeling methods. In particular, we train a model to jointly predict surface normals and semantic labels, using a joint network with two small output decoder heads but a shared encoder for both tasks. We will show how the joint encoder-decoder improves both surface normal estimation and semantic labeling results as opposed to learned alone. In the next chapter, We will show that learning the surface normals gives better results than learning raw depth.

Finally, since most of the applications of surface normal estimation require a real-time method, we propose some simple tricks to make our model much smaller and faster, with negligible loss in accuracy.

In summary, our primary contributions in this chapter are as follows:

- A method for computing reliable ground truth normals using "semantic smoothing".
- A method for training with synthetic and real data which gives state of the art results.
- A method that jointly learns semantics and surface normals in an end-to-end manner, increasing the accuracy of both.
- A novel joint encoder-decoder architecture.
- A way to make the model run in near real-time on a mobile phone while still giving good accuracy.

7.2 Method

7.2.1 Computing better ground truth normals

In this section, we discuss two simple methods for computing better ground truth normals from (real) depth datasets.

7.2.2 Datasets

We use two commonly used real-world depth datasets, NYUDv2 and Scannet, both described in Section 2.2. Examples of our predictions on NYUDv2 are shown in Figure 7.2

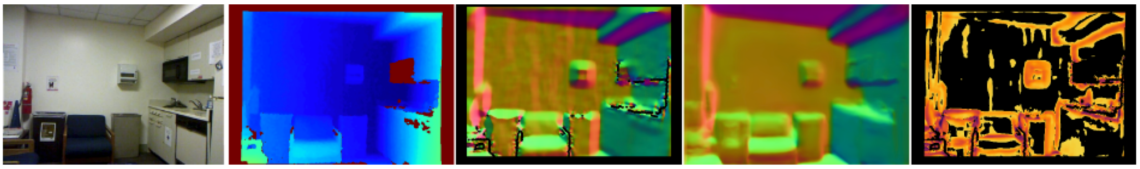


Figure 7.2: Illustration of the NYU data, together with the predictions of our model on them. Columns, from left to right: RGB image, depth, ground truth surface normals, our predictions, error image (where black is under 11.25 degrees errors, and then error increases from yellow to purple).

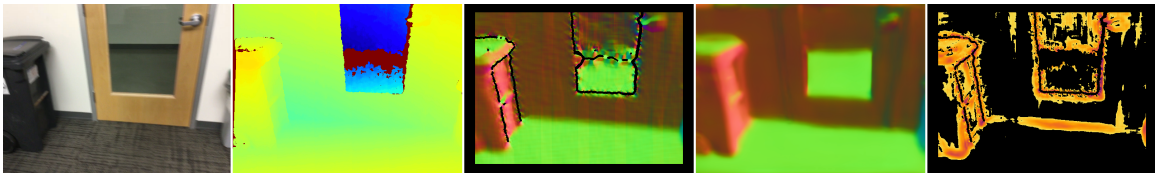


Figure 7.3: Qualitative evaluation on the Scannet data. Columns are the same as in Figure 7.2.

and examples on Scannet are shown in Figure 8.2.

7.2.3 Problems with current techniques

Surface normals for real world datasets are typically derived from the depth data captured by commodity depth sensors or stereo matching algorithms. For instance, the NYUDv2 dataset was captured using a Kinect v1, while Scannet uses a similar Structure sensor. These sensors are well known to suffer from axial noise which is related to the distance of the surface from the sensor. As a consequence, surface normals that are computed from this data tend to exhibit artifacts that are noticeable, especially on distant planar regions.

Broadly speaking, prior work has used one of two normal estimation methods to generate ground truth for training: least-squares estimation on a per-point basis using RANSAC after denoising the point cloud [86], or local plane computation using the covariance matrix over a window [15]. In our experiments, we have found both of these approaches result in ground truth normals that have considerably more errors. For instance, in Figure 7.4, the former method produces oversmoothed and incorrectly oriented planar patches on regions like the sink, while in Figure 7.5(d), the latter method produces highly noisy planar

Figure 7.4: A common example of the errors seen in the normals from [86] that many use for training and evaluation. In these visualizations, (r, g, b) map to (x, y, z) of the normal at that location. Note the over-smoothing, which reduces and removes the normals of small objects. This image also demonstrates why it is important to only backpropagate on pixels that have valid depth, as the right side of the image has incorrect normal data due to noisy and missing depth values.

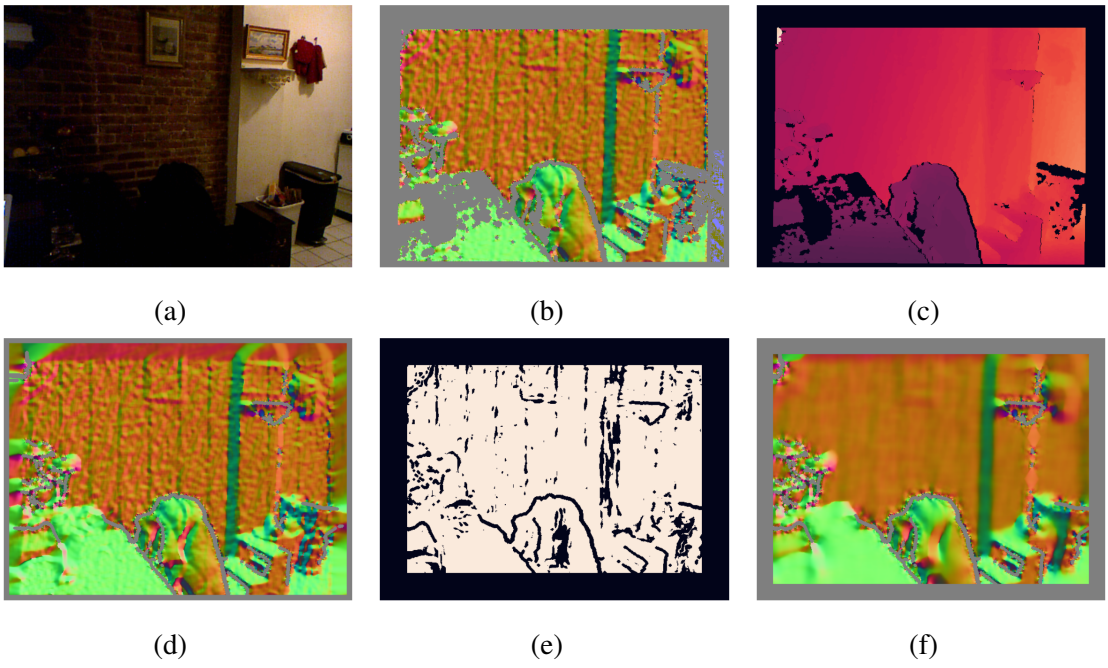


Figure 7.5: Effect of window smoothing size on surface normals. a) RGB image. b) Normals computed with a normal smoothing size of 10. c) In-painted depth. d) Normals computed similar to [15] from (c) using window size 10. e) The mask generated for valid training pixels. f) The normals with window size 30 that we train on (backpropagating only when e is valid).

surfaces. We hypothesize that this could cause inferior results when used to train. In Table 7.1, we show that training on noisy ground truth normals results in a noticeable drop in accuracy. Training on data using the smaller smoothing window of [15] results in a mean angle error of 27.5 degrees, compared to 22 degrees when trained on our proposed normals. When visual inspecting the results shown in Figure 7.5 and Figure 7.4, it is obvious that many past methods have been training and evaluating on erroneous data. An easy way to see this is to check for color changes on planar surfaces such as walls.

7.2.4 Improving Normals Using Point Clouds

We propose to use the method introduced in [128] to compute surface normals from a point cloud. We begin by smoothing the depth and filling holes as in [89] and then construct a 3D point cloud with PCL [135]. A key insight here is to use a large smoothing window that adequately compensates for the noise introduced by the sensor, while not smoothing over visually salient depth discontinuities. While this is a straight-forward approach, it is important to note that it has not been done by any of the previous work and our ablation studies show it greatly improves results. For this, we use the integral image approach implemented in PCL [128]. Compared to [15], this samples a larger window more densely without incurring a disproportionate computational cost. We select a smoothing radius of 30 for both real datasets (NYUDv2 and Scannet) and 10 for the synthetic Scenenet dataset since it has minimal noise in its rendered depth estimates. We evaluate this method in Section 7.2.6.

7.2.5 Improving Normals with Semantics

To further reduce the amount of noise and get closer to absolute truth, we can leverage the semantics of the datasets. For certain semantic classes, e.g. walls and floors, we know that the results are usually planar (or at least piece-wise planar). We use that information to smooth out the normals for those instances. Given that the datasets we are using all have

some level of semantic information labeled, this is free contextual information.

While semantic segmentation gives us labels for objects, it does not distinguish between facets of the same class (for instance, walls facing in different directions). We perform an efficient post-processing step to identify regions with pixels that have consistent normals and semantic labels. We adapt the standard connected components algorithm to start at a pixel and grow the region outwards by adding pixels with normals that are within 30 degrees of the current averaged normal of the region, and of the same class as the starting pixel. We restrict this process to semantic labels that we assume to be planar. However, even if this assumption is violated, the normal variance constraint prevents arbitrary growth of these supposedly planar regions. Once we have computed the regions, we assign the averaged normal to all pixels of this region if the region is of a minimum size. An example of this is shown in Figure 7.1. We evaluate this method in Section 7.2.6.

7.2.6 Evaluation

Table 7.1: Accuracy and error rates of our normals model (without semantic output head) when evaluated on the NYUDv2 normals from [86]. First row shows results using our simple network training on standard normals as shown in the top right of Figure 7.1. Second row shows results using our denoising method (bottom left of Figure 7.1). Third row shows results using our semantic smoothing method (bottom right of Figure 7.1). Evaluation is performed on the normals from [86], which demonstrates the necessity of a dataset cleanup.

	Accuracy			Error
Method	$\leq 11.25^\circ$	$\leq 22.5^\circ$	$\leq 30^\circ$	Mean Angle
Baseline	46.2%	57.7%	63.8%	27.5°
Denoising	49.5%	64.6%	71.1%	22°
Semantics	**60.6%**	**77.9%**	**83.4%**	**14.7°**

Table 7.1 shows an ablation study done on NYUDv2 with different types of training data all evaluated on [86] with the simple mobile encoder-decoder network described in Section 7.5 without the performance increases we discuss later. The first row shows the

results when trained on the normals used by many methods as shown in the top right of Figure 7.1, the second row shows training on our normals computed from Section 7.2.4, the final row shows training on our proposed ground truth surface normals that are semantically corrected. The normals from Section 7.2.4 result in much better accuracy compared to [89] or [86], especially in the larger angle errors and the mean angle error. Leveraging semantics improves results even more. This simple idea improves the mean angle error by almost 13 degrees and reduces the smallest angle errors by 14%. This is a substantial increase better than most new architectures would yield.

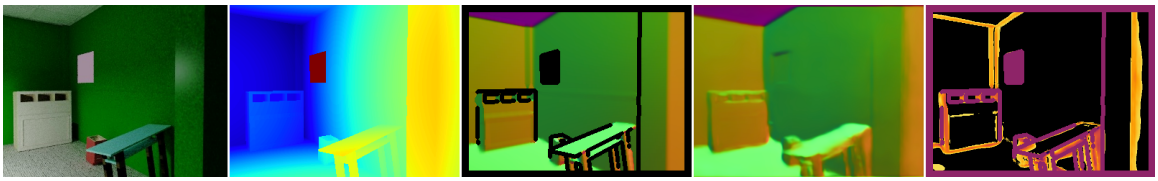


Figure 7.6: Qualitative evaluation on Scenenet data. Columns, from left to right: RGB image, depth, ground truth surface normals, our predictions, error image (where black is under 11.25 degrees errors, and then error increases from yellow to purple).

7.3 Combining synthetic and real data

We train and evaluate our network on several publicly available datasets, both real and synthetic, to reduce our dataset bias and produce more robust normals, as we explain below. For synthetic, we use Scenenet RGBD. For the real datasets, we use Scannet and NYUDv2. All 3 are described in Section 2.2. We compute the normals for Scenenet using the method proposed in Section 7.2.1; however, we use a normal smoothing size of 10 given the input depth data is less noisy than data from conventional depth sensors.

7.3.1 Mixing real and synthetic

Since most prior work utilizes NYUDv2, our method was initially trained and evaluated only on it. However, we found this doesn't necessarily generalize well to other data, as shown in Table A.1. We found that fine-tuning from synthetic to real was actually sub-

optimal compared to our batch-wise mixing strategy. Results on all datasets are shown in Appendix A. (All results are obtained using the normals branch of the model architecture that is shown in Figure 7.9; see Section 7.5 for details.)

We also discovered that the standard practice of pre-training on Scenenet and fine-tuning on NYU results in a model that generalizes poorly. However, by simply mixing 10 synthetic Scenenet images with 1 real image in every mini-batch, we were able to improve performance on both datasets. Best results were obtained by mixing all 3 datasets, using 10 parts of Scenenet, 5 parts of Scannet, and 1 part of NYUDv2. Qualitative results on Scenenet are shown in Figure 7.6, on NYU are shown in Figure 7.2, and on Scannet in Figure 8.2.

Table 7.2: Comparison against state of the art on surface normal estimation task. All methods (except in the last row) are evaluated on the normals from the NYUDv2 dataset computed using the method of [86]; in the last row, we show our method evaluated on the normals from NYU computed using our method, which we denote by NYU'.

	Accuracy			Error	
Method	11.25	22.5	30	Mean Angle	rmse
Fouhey[141]	39.2	52.9	57.8	35.2	-
Zeis[86]	27.7	49.0	58.7	33.5	-
Dharmasiri[109]	44.9	67.7	76.3	20.6	-
Wang[90]	47.3	68.9	76.6	20.6	-
Qi[112]	48.4	71.5	79.5	19	26.9
Ours	48.9	**72.3**	**81.2**	**17**	30.2
Ours on NYU'	**59.5**	**72.2**	77.3	19.7	**19.3**

7.3.2 Comparison with the state-of-the-art

In Table 7.2, we show our results compared to previous state of the art surface normal estimation methods. Previous methods compute normals using the method of [15] or [86] applied to various datasets. Here we compare all methods by evaluating on NYUDv2; for baseline methods, we compute normals using the method of [86], whereas for our method,

Table 7.3: Normal Accuracy comparisons with different testing and training datasets. The columns are the training set used and the rows are the evaluation accuracy for each individual dataset. Scenenet+NYU FT means the standard practice of pretrained on synthetic and finetuned on NYUDv2. Datasets Mixed means the dataset is trained all from scratch with a batch-wise mix. The best result for each row is bold.

	Training set	
Accuracy	Scenenet+NYU FT	Datasets Mixed
NYU		
$\% < 11.25$	57	**59.5**
$\% < 22.5$	69.6	**72.2**
$\% < 30$	74.6	**77.3**
Mean Angle Error	21.3	**19.7**
Scannet		
$\% < 11.25$	36.7	**50.1**
$\% < 22.5$	54.6	**63.2**
$\% < 30$	60.9	**68.2**
Mean Angle Error	34.1	**28.8**
Scenenet		
$\% < 11.25$	21	**64.5**
$\% < 22.5$	48.2	**70.7**
$\% < 30$	59.9	**68.2**
Mean Angle Error	37.6	**26.1**

we use our approach for computing normals during training. For testing, all methods use the method of [86] to compute normals. We outperform the previous state-of-the-art (SOTA), [112], despite using a more than 100x smaller model, due to the higher quality of our data. For completeness, we also report the performance of our method when evaluated on our proposed method of computing normals from NYU; this test set is more accurate, and is also more similar to training, so we see performance is even greater.

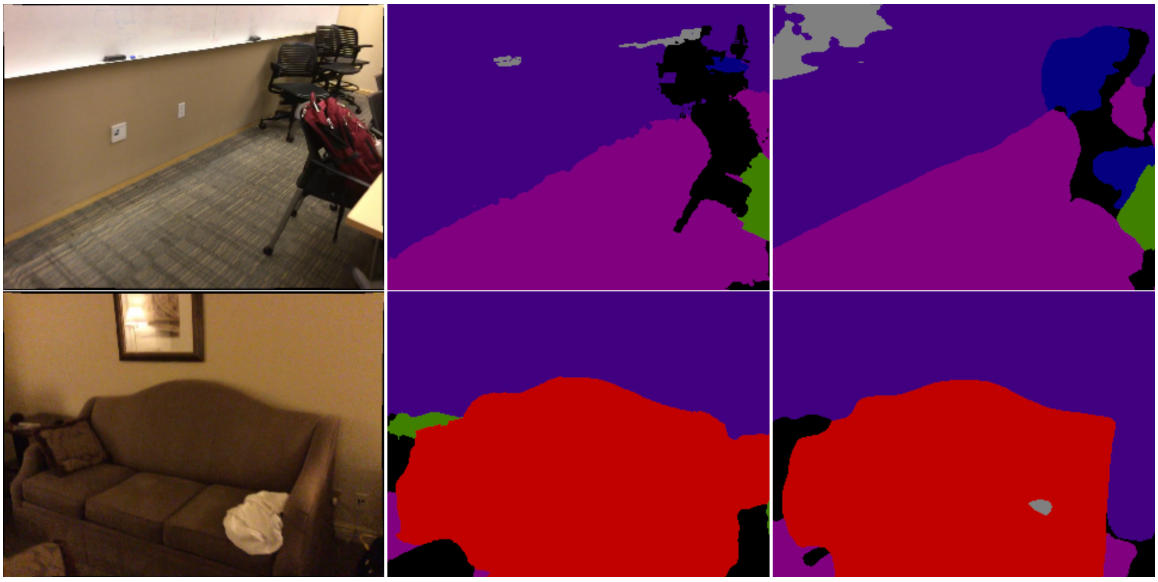


Figure 7.7: Examples of our semantic labeling predictions for the Scannet dataset. From left to right: RGB image, ground truth, our predictions.

7.4 Jointly predicting semantics and normals

In order to evaluate the effect of combining normals and semantic labeling, we did an ablation study using just the Scannet dataset. We chose it because it is a large, complex, real dataset with both normals and semantic labels. We used the 13 semantic labels from NYUDv2 for our experiments. These consist of bed, books, ceiling, chair, floor, furniture, objects, picture, sofa, table, tv, window, and wall. To fairly compare, we train on only Scannet in this ablation study, and ignore other datasets.

7.4.1 Semantics

In order to train semantic labeling, we use our same architecture and training with slight changes. We change the regression output of 3 channels with a cosine loss to a classification output of 14 labels using a softmax cross-entropy loss. Experimentally, semantic labeling seems to be a harder task than surface normal estimation, so in order to train, we fine-tune our whole architecture initialized from our normals prediction with a lower learning rate (0.001). Semantic only prediction is shown in the Semantics column of Table 7.6. See

Figure 7.8: Examples of our semantic labeling predictions for the Scenenet dataset. From left to right: RGB image, ground truth, our predictions.

Figure 7.7 for some qualitative results. Note that there is occasional error in the ground truth of the semantics as well. Even though semantics are just an intermediate task for our method, our results are still very promising. Our prediction in Figure 7.7 correctly predicts both chairs as chairs (blue), even though the ground truth doesn't have this labeled correctly.

7.4.2 Joint Prediction

To train our method jointly, we duplicate the decoder using the architecture shown in Figure 7.9. We then fine-tune both the encoder and the dual decoders using the weights from our normal prediction network. The cosine and softmax cross-entropy losses are summed with a 20x weight modifier given to the cosine loss to balance them. An ablation study

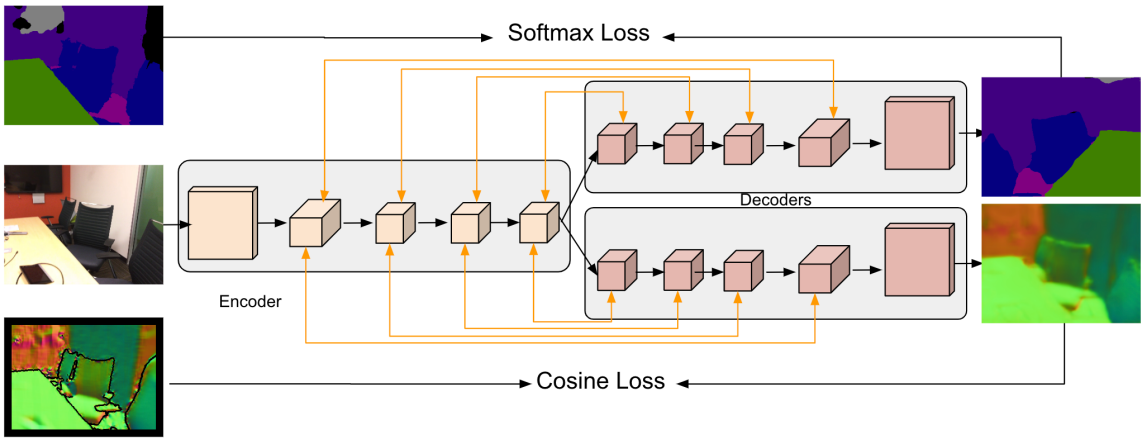


Figure 7.9: Our Architecture for joint prediction involves a shared encoder inspired by Mobilenet [142], followed by two U-net decoders. Each outputs its prediction and has a separate loss for either segmentation or normal prediction. The losses along with regularization are summed and optimized jointly. When doing just normal prediction, we simply drop the segmentation decoder and loss. See Section 7.5 for details.

showing this is in Table 8.7. Given that the cosine loss of normal prediction is about 10-20x lower than the softmax cross entropy of semantic labeling, weighting this loss makes sense. Note, that for our network, weighting the loss 20x seems to produce the highest results.

Table 7.4: This shows the effect of multiplying the cosine loss learning rate.

	Normals			Semantics
Method	$\leq 11.25°$	$\leq 22.5°$	$\leq 30°$	Pixel Accuracy
LR Mult 1	58	69.7	72.5	88.1
LR Mult 5	61.5	70.8	73.2	88.7
LR Mult 10	63.1	71.4	73.5	88.7
LR Mult 20	**64.4**	**72.3**	**74.2**	**89.7**

Contrary to prior work[111] that shows joint prediction reduces accuracy, our network improves when combining both semantics and normals. This is shown in Table 7.6. Surface normal estimation improves slightly (though it is important to note that small changes in the surface normal accuracy can still make large differences in practice due to the difference in

angle error being so noticeable when wrong). Interestingly, semantic labeling gets a large 6% increase in pixel accuracy. We hypothesize this is due to the importance of shape as a cue for semantic labels as pointed out in our Thesis Statement. Note that this actually outperforms the previous results evaluated on Scannet in Table A.1 as well. It's possible that without new normals proposed in Section 7.2.1, this performance increase would not happen.

Table 7.5: This table shows the different effect of the learning rates of the semantic trained network.

Method	Pixel Accuracy
Trained from Scratch:	
LR 0.045	45.6
LR 0.01	45.6
LR 0.001	80.2
Fine-tuned:	
LR 0.001	85.8

For training the semantic network, we also need to use a lower learning rate than our surface normal network. This is demonstrated in Table 7.5, which shows how varying the learning rate changes the efficacy of the network when learning semantics as opposed to normals. Note that though normals learn with a learning rate of 0.045, that is too high for semantics. This is potentially due to it being a much harder problem with 14 classes as opposed to 3. When reduced to 0.001, this accuracy drastically increases. When finetuned from the normals network, the accuracy increases even more. This is why we fine-tune our semantics network with a lower learning rate. We treat the joint network in the same manner.

7.5 Training a real-time model

In this section, we describe how to use the above techniques, combined with a lightweight model, to build a real-time mobile system with state of the art accuracy. We discuss our

Table 7.6: This shows the results of Joint semantics and normals prediction on Scannet. The Normals column is the accuracy of a network only trained on Scannet normals. The semantics is the accuracy when only trained on Scannet semantics. Joint is the accuracy when both are trained concurrently as per Section 7.4.

	Method		
Accuracy	Normals	Semantics	Joint
$\% < 11.25$	49.3	N/A	**50.9**
$\% < 22.5$	63.2	N/A	**65.2**
$\% < 30$	68.2	N/A	**70**
Mean Angle Error	29	N/A	**28**
Semantic Accuracy %	N/A	59	**65.6**

model size, training pipeline, and important tricks (reducing the number of channels and utilizing grayscale) used to get the model on a mobile device.

7.5.1 Model

Prior approaches to the task of normal prediction have used feature extractors trained on VGG [143] or ResNet [70]. In contrast, we use a light-weight architecture that lends itself well to mobile applications. For our surface normal experiments and ablation studies, we use a modified version MobileNetV2 [142] encoder followed by the U-net decoder [144]. The key changes to MobileNetV2 are a normal residual instead of the inverted residual, PReLU [145] instead of ReLU, removing global average pooling, and increasing the convolutional filter size to 5.

For the decoder, we use U-net with 4 bilinear resizes, convolutions, and concatenations. These correspond to the blocks in MobileNetV2. The final output of the decoder is resized to the width and height of the input image (320 x 240 in our experiments), with the number of channels defined by the output task (i.e. 3 for normals and 14 for semantic labeling of NYU13).

After training our network, we can remove unnecessary ops and only use the normals

encoder-decoder path by converting that model to a flat-buffer using Tensorflow Lite[146]. Our final network is under 2MB in size. We run inference using this model on the phone via ops implemented as OpenGL shaders.

7.5.2 Finetuning vs. Training from scratch

Conventionally, encoder-decoder networks use a larger encoder like ResNet101 (which is pretrained on Imagenet) and then fine-tune them for the specific task. However, for the task of surface normal estimation, we found that training from scratch in an end-to-end manner gave us better results. This could be due to the Imagenet dataset bias, our small network encoder, or the uniqueness of the task.

For training, when learning only surface normals in a single architecture as in our ablation studies, we use RMSProp [147] with a weight decay of 0.98, a learning rate of 0.045. When we train on surface normals and semantics, we fine tune off the surface normals model with a lower learning rate of 0.001.

7.5.3 RGB vs. Grayscale

For the task of surface normal estimation, color doesn't give much more of an advantage over grayscale data. This is shown in Table 7.7. This suggests that the neural network learns edges and color invariant features. This can potentially reduce the size and number of operations in a network. An ablation study on this is shown in Table 7.7.

7.5.4 Network Size

In Mobilenet [142], an encoder-decoder architecture is proposed with network size and speed described in the number of multiply-adds (MpAdds). We also test our normal prediction network as a function of network size in the same manner. The results are shown in Table 7.8.

A semantic segmentation model was also proposed by [142] with DeepLab as a de-

Table 7.7: This ablation study shows the effect of color on the network by changing the percent of input training images that are converted to grayscale. Interestingly enough, color does not seem that important for surface normal estimation.

	% of images grayscale		
Metrics	0%	50%	100%
NYU			
$\% < 11.25$	59.1	59.2	59.1
$\% < 22.5$	72.2	72.2	72.2
$\% < 30$	77.4	77.3	77.3
Mean Angle Error	19.5	19.4	19.5
Scannet			
$\% < 11.25$	49.6	49.5	49.6
$\% < 22.5$	63.6	63.6	63.4
$\% < 30$	68.6	68.6	68.5
Mean Angle Error	28.8	28.8	28.9
Scenenet			
$\% < 11.25$	60.7	63.2	63.3
$\% < 22.5$	70.3	70.6	70.5
$\% < 30$	72.9	72.9	72.8
Mean Angle Error	27.1	26.5	26.5
Average			
$\% < 11.25$	57.075	57.625	57.7
$\% < 22.5$	69.025	69.05	68.975
$\% < 30$	73.325	73.3	73.225
Mean Angle Error	24.225	24.075	24.125

coder, where the last encoder layer is removed to minimize model size. The deeplab model is 2.75B MpAdds with stride 16 and 152.6B MpAdds with stride 8. We found that it is actually better to keep the last layer and just reduce the channel size of each layer; this results in a faster and smaller model. Our proposed network has 1.624B MpAdds at its largest, and only 467M for the mobile version, 300x smaller than the fast deeplab version. The other SOTA methods we compare against earlier in the chapter, that utilize Resnet-101 or VGG-19, have between 91B and 5000B MpAdds, which is several orders of magnitude larger than our proposed network even though our method has much results, which is a large contribution.

Table 7.8: Here we use an ablation study to test performance vs network size. The channel multiplier is a multiplier that determines the number of output channels calculated at each block. For instance, the final output of the encoder at channel multiplier 32 has 1280 channels, whereas, at channel multiplier 16, it has 640 channels.

	Channel Multiplier			
Accuracy	12	16	22	32
NYU				
$\% < 11.25$	56.1	57.1	58	59.3
$\% < 22.5$	67.7	68.6	69.3	69.6
$\% < 30$	72.3	73.1	73.7	73.9
Mean Angle Error	22.8	22.3	22	21.8
Scannet				
$\% < 11.25$	44.5	46	47.6	50.1
$\% < 22.5$	60.3	61.4	62.3	63.2
$\% < 30$	65.9	66.9	67.6	68.2
Mean Angle Error	30.6	30	29.5	28.8
Scenenet				
$\% < 11.25$	59.9	61.5	62.3	64.5
$\% < 22.5$	68.8	69.6	70	70.7
$\% < 30$	71.4	72.1	72.4	78.2
Mean Angle Error	27.8	27.2	26.9	26.1
Average Eval				
$\% < 11.25$	54.08	55.13	56.48	58.15
$\% < 22.5$	66.55	67.35	68.20	68.63
$\% < 30$	71.10	71.85	72.53	74.10
Mean Angle Error	25.55	25.05	24.63	24.20
# million MpAdds	467	673	987	1624

7.5.5 Applications

Using the channel scaling and other model minimization techniques discussed above, we created a lightweight architecture that runs at 12fps on a mobile phone. To demonstrate our SOTA results on normal estimation in real-time, we use this estimation to place stickers on surfaces in their natural orientation. Screenshots showing this demo running on the mobile device are shown in Figure 7.10 and a video is shown in [140]. By simply clicking a region, the sticker or object can be placed realistically in AR using the predicted normals.

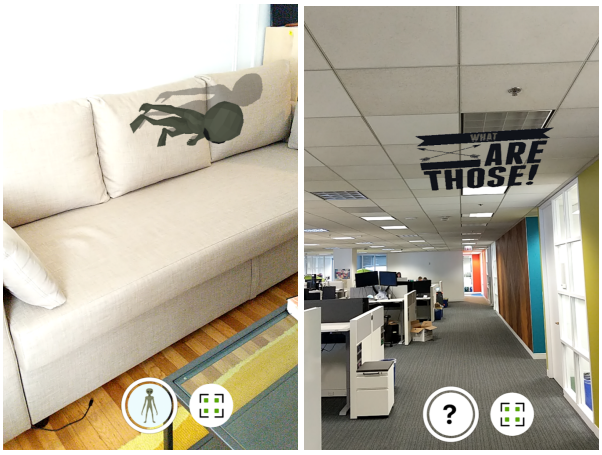


Figure 7.10: A sample AR application that uses the surface orientation to place a virtual character and text.

7.6 Conclusion

In this chapter, we have shown several methods for significantly improving the accuracy of any CNN method for predicted surface normals, namely: calculate the ground truth normals in a better way; combine real and synthetic data in a better way; and jointly train for normal prediction and semantic segmentation. We have also shown how to use these ideas to train a lightweight model that gives state of the art results, has low memory footprint, and runs at interactive rates on a mobile phone. This is yet another method in which we combine semantics and shape in order to improve our results.

In the next chapter, we will propose modifying this network architecture both decreasing it's number of parameters and increasing it's predictive accuracy by investigating different network fissions (i.e. where you split the decoder into separate outputs). We will also investigate the relationship between depth, surface normals, and semantic labels.

CHAPTER 8

SHARING DECODERS: NETWORK FISSION FOR MULTI-TASK PIXEL PREDICTION

In this chapter, we continue exploring where we left off in the last chapter, multi-task learning. In the last chapter, we proposed an encoder-decoder method to predict both semantic labels and surface normals. However, we didn't discuss much about what the best way of going about that is. Which part of the encoder-decoder network should you split from when doing one-to-many prediction? The convention of hard parameter sharing is after the end of the encoder, with a decoder for each task, but is that best?

In this chapter, we do several ablation studies to explore this problem for the tasks of semantic labeling, surface normal estimation, and depth estimation. We examine the benefits of splitting encoder-decoder networks and sharing the decoder for multi-task labeling. We introduce a multi-task pixel-wise labeling method that improves results on several tasks without adding many parameters. All current hard parameter sharing methods for multi-task pixel-wise labeling use a simple scheme of one shared encoder with separate decoders for each task. We do several ablation studies on three different tasks (semantics, surface normals, and depth), splitting at different places in encoder-decoder architectures, which we call fission (where the task splits into its own separate layers for output). These studies show that sharing most of the decoder layers in multi-task encoder-decoder networks results in improvement on several tasks compared to current methods, which split at the encoder. In addition, sharing most of the decoder layers requires very few extra parameters per task.

Our proposed method trains faster, uses less memory, results in better accuracy, and uses significantly fewer floating point operations (FLOPS) than conventional multi-task methods, with additional tasks only requiring 0.017% more FLOPS than the single-task

network. We demonstrate state-of-the-art results on 3 datasets for surface normal estimation and semantic labeling.

Experiments demonstrate that combining surface normals with semantic labels results in better semantic labels than combining depth with semantic labels and do several ablation studies showing that a network trained on surface normals is a better initialization than one trained on depth, regardless of the tasks chosen. We also show that both predictions improve and that fewer extra parameters are required. These are all three major points discussed in our Thesis Statement 1.1.

8.1 Introduction

Multi-task learning aims to jointly learn multiple tasks and helps with generalization, improving results given an inductive bias[102]. Deep learning methods have specifically made great improvements in multi-task pixel-wise prediction tasks using architectural improvements of encoder-decoders[148]. Early work by Caruana[99] shows that hard parameter sharing with backprop nets "discovers task relatedness without the need of supervisory signals"; however, this hasn't been extended fully for encoder-decoders. Motivated by this, we undertake an exploration of hard parameter sharing of encoder-decoders with several outputs, which we call *network fission*. We consider (a) early fission (splitting the network at the encoder with separate decoders, Figure 8.1a), (b) late fission (splitting the network only at the very end and sharing the entire encoder-decoder Figure 8.1d), and (c) mid fission (sharing most of the encoder-decoder and splitting at one of the intermediate decoder blocks, Figures 8.1b and 8.1c).

Initial work for semantic labeling and depth prediction focused on using fully convolutional networks (FCNs) [149] but later decoder-based architectures were proposed [150, 151] to overcome the limitations of FCNs. Encoder-Decoder architectures were shown to be very effective and soon became the established method for pixel-wise prediction tasks. When doing multitask pixel-wise predictions, hard parameter sharing is the most popular

method and it is common to take the features at the end of the encoder and then split to give each task its own decoder. This is the current preferred method used by many [111, 114, 115, 116, 6], which we refer to as early fission.

There are several recent soft parameter sharing methods[101, 102, 152] that have a model for each task and learn parameters between single task networks to combine, stitch, and/or regularize them; however these can be difficult to learn and are prone to over-fitting. In terms of hard parameter sharing, currently, most methods only use early fission (splitting at the encoder bottleneck with separate decoders for each task) as shown in Figure 8.1a. We do several ablation studies on different fission methods and find that mid fission (sharing all of the decoder blocks until the last decoder layer) outperforms currently used multi-task methods and requires significantly less FLOPS. We evaluate our method on the tasks of depth, surface normals, and semantic labels.

The main goal of this chapter is to examine the benefits of splitting encoder-decoders in different places for multi-task learning and showcase improvements using new frameworks. In the past, researchers explored combining data with different input fusion schemes; similarly, we should also examine different output fission schemes. For comparison, an example of early fusion [153, 78] is where RGB and depth are stacked as 4 channels and input into the same set of convolutions. Late fusion [154, 8] for the same task puts each input modality into a set of convolutions and joins the outputs at the end. Work later proposed mid-fusion [155] where each input modality has some encoding, but then those features are concatenated and further convolutions are done before an embedding is reached. We can make the same deductions for different outputs and explore early, mid, and late fission.

An easy mapping can be made from these modality fusion schemes to our proposed modality fission schemes. Early fission as shown in Figure 8.1a is the current method of splitting at the encoder bottleneck and giving each task its own decoder. Late fission as shown in Figure 8.1d would be both the encoder and decoder fully shared with only one

output, computing a loss differently on different channel subsets of the output. For the commonly used 3-stage decoder blocks, there are two potential splits that qualify as mid fission. Figure 8.1b shows what we call early-mid fission: sharing all of the encoder and the first decoder block. Figure 8.1c shows what we call mid fission: sharing all of the encoder and most of the decoder but allowing the last set of layers to learn features separately for each output modality. The proposed mid fission method has not been explored and analyzed before and many multi-task methods would benefit from this architectural change.

In summary, our primary contributions in this chapter are as follows:

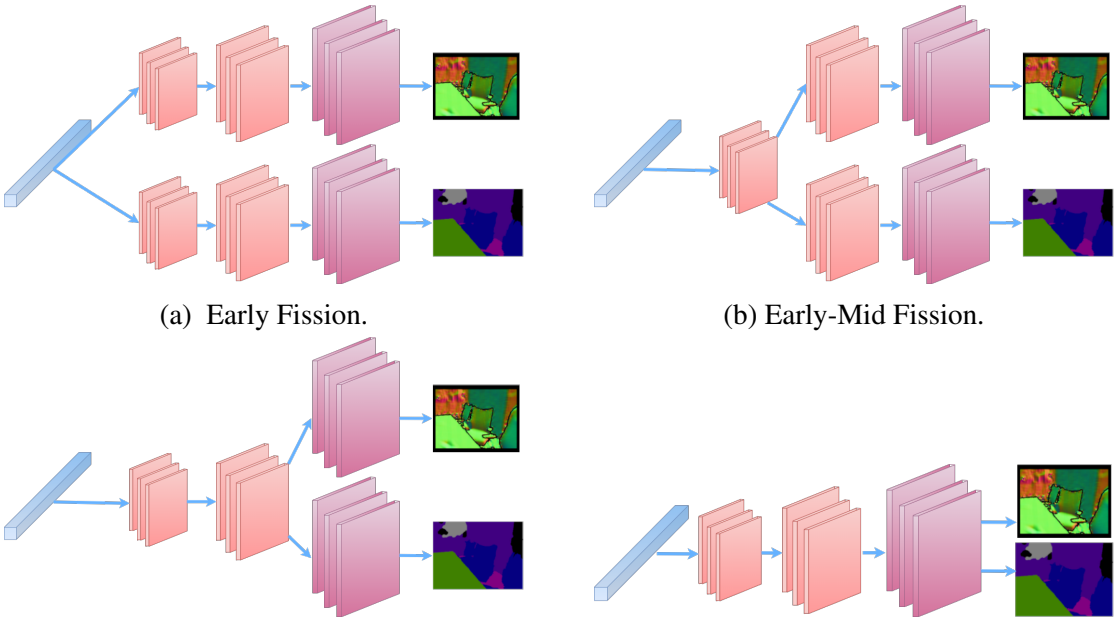
- Comprehensive ablation studies comparing fission schemes with several modalities.
- A new shared decoder architecture we call mid fission that has fewer parameters, outperforms single task training, and outperforms current multi-task methods with significantly less FLOPS.
- State of the art results on 3 different challenging datasets on several modalities.
- A method that learns other tasks with only 0.0167% more FLOPS required per task.
- Code released[156] with all of the parameters and details for every ablation study.

8.2 Method

Here, we detail the proposed method for studying the different fission schemes. The backbone architecture and hyperparameters should remain the same for all ablations in order to have a valid comparison. We'll explain the proposed architecture, the losses, and each method (early, late, and mid fission).

8.2.1 Backbone Architecture

For the backbone of all of our ablation studies, we adopted the Adapnet++ encoder-decoder architecture from [157] (an Adapnet improvement). This architecture was chosen because



(a) Early Fission. (b) Early-Mid Fission.

(c) Mid Fission. (d) Late Fission.

Figure 8.1: The possible fissions for the eASPP decoder part of the architecture[157]. The blue block is the encoder bottleneck, the orange blocks are the first two decoder blocks, which include convolutional, batchnorm, and relu layers with 256 output channels and skip connections. The purple blocks are the last decoder block which has convolutional transpose, batchnorm, and relu layers.

it has state of the art results on several datasets despite requiring fewer parameters. Importantly, it does not include depth or other additional inputs as we are exploring RGB-to-many prediction. We also use two auxiliary loss branches at the end of the first and second stages of the deconvolutional layers just as in [157]. Each branch decreases the features to the number of labels using 1x1 convolution with batch normalization and then bilinear upsampling to match the input resolution. We consider the tasks of semantic labeling, depth prediction, and surface normal prediction. For all equations below, α_1 is 0.6, and α_2 is 0.5 for the auxiliary loss balancing. The encoder is ResNet-50[145] with full pre-activation residual units[70] and multi-scale residual units [157] with varying dilation rates. After the encoder, we use the efficient atrous spatial pyramid pooling module (eASPP) [157] to create the bottleneck. The output of this is 16-times down-sampled.

For the decoder, we use three stages comprising of convolutional and deconvolutional

layers. The first stage is upsampled by a factor of 2. The second stage concatenates those results with the first skip refinement from the encoder with a 1x1 convolution. That result is passed through two 3x3 convolutions followed by a deconvoluional layer that upsamples by a factor of 2. The second stage is the same as the second stage except using the 2nd skip refinement from the encoder. The output is then finally used as the input to a 1x1 convolutional transpose to reduce the number of feature channels to the desired output for the task which is then upsampled 4x to match the input resolution. This is better seen in code[156]. We find for multi-task learning, the auxiliary branch losses help improve results, discussed in Section 8.3.5.

8.2.2 Losses

For semantic labeling, we use the softmax cross entropy loss with auxiliary losses in Equation 8.1, where y is the one-hot encoded ground truth pixel-wise label, σ is the softmax function, z is the final output of the task decoder, z_{aux1} and z_{aux2} are auxiliary branches of the decoder as in [157]. ϵ is 1^{-10} for stability.

$$
\begin{aligned}
L_{\text{semantics}}(y, z) = & \\
& - \sum_{i=0}^{n} y^{(i)} \log(\sigma(z^{(i)}) + \epsilon) - \alpha_1 \sum_{i=0}^{n} y^{(i)} \log(\sigma(z_{\text{aux1}}^{(i)}) + \epsilon) \\
& - \alpha_2 \sum_{i=0}^{n} y^{(i)} \log(\sigma(z_{\text{aux2}}^{(i)}) + \epsilon)
\end{aligned}
\tag{8.1}
$$

For surface normal prediction, we use the cosine similarity with auxiliary losses in Equation 8.2. In this equation, $\hat{y}$ is the pixel-wise ground truth surface normals normalized to a unit vector where each normal is clipped between -1.0 and 1.0 to prevent numerical errors. $\hat{z}$, $\hat{z}_{aux1}$, $\hat{z}_{aux2}$ are the final outputs of the task decoder and auxiliary branches all normalized and clipped the same as $\hat{y}$.

$$L_{\text{normals}}(y, z) = 1 - \sum_{i=0}^{3} \hat{y}^{(i)} \hat{z}^{(i)} + \alpha_1 \left(1 - \sum_{i=0}^{3} \hat{y}^{(i)} \hat{z}^{(i)}_{\text{aux1}} \right) + \alpha_2 \left(1 - \sum_{i=0}^{3} \hat{y}^{(i)} \hat{z}^{(i)}_{\text{aux2}} \right) \quad (8.2)$$

For depth prediction, we use the L1 loss with auxiliary losses in Equation 8.3 where y is the ground truth pixel-wise depth divided by 1000.0 so as to be in meters. z, z_{aux1}, z_{aux2} are the final outputs of the task decoder, the first auxiliary branch, and the second auxiliary branch respectively.

$$L_{\text{depth}}(y, z) = |y - z| + \alpha_1 |y - z_{\text{aux1}}| + \alpha_2 |y - z_{\text{aux2}}| \quad (8.3)$$

Now we discuss the training used for all the proposed early, mid, and late fission schemes.

8.2.3 Early Fission

As discussed in Section 3, early fission is a shared encoder with separate decoders for each task shown in Figure 8.1a. This method has been widely used in the literature. The bottleneck from the encoder connects to two separate decoders with their own skip connections and deconvolutional layers. The loss is then computed via Equation 8.4 where L_T is the total loss. For each task 1 to n there is an individual loss L_i and a loss weight λ_i. In much work, there is no loss balancing, meaning all λ is set to 1.0, in others it's set manually based off of loss convergence properties[6], and others learn it[114]. In our experiments, when we balance the losses, we set $\lambda_{\text{semantics}}$ to 1.0, λ_{normals} to 10.0 and λ_{depth} to 0.5 based off of the loss values after 10K iterations when the network starts to converge.

$$L_T = \sum_{i=1}^{n} \lambda_i L_i(y_i, z_i) \quad (8.4)$$

8.2.4 Early-Mid and Mid Fission

In our proposed shared decoder architecture, we use mid-fission as shown in Figure 8.1b and Figure 8.1c. The loss for this is calculated the same as in Section 8.2.3. The only difference is where we split the decoder. We found both early-mid and mid fission to have superior results, shown in Table 8.2.

8.2.5 Late Fission

In late fission, we explore what happens if the entire decoder and encoder are shared as shown in Figure 8.1d. The output of this is treated as a concatenated set of task-level outputs with the loss computed via Equation 8.5. Here there are several loss equations L_i and several labels y_i but only one output z, which is partitioned into n splits of size τ_i depending on the output size. In our case, $\tau_{\text{semantics}}$ is the number of semantic labels plus one (for the background class), τ_{normals} is 3, and τ_{depth} is 1. λ is the same as in early fission.

$$L_T = \sum_{i=1,k=0}^{n,k+=\tau_i} \lambda_i L_i(y_i, z[k : (k + \tau_i)]) \tag{8.5}$$

8.2.6 Relationship to State-of-the-art Methods

Our method is a type of hard parameter sharing, meaning sharing hidden layers between tasks followed by task-specific output layers[102]. Here the task-specific and task-shared features are learned in the hidden layers but everything else is fixed. This has the benefit of being simple to learn and reducing over-fitting as discussed in[99, 152] with the detriment of not being robust to loosely/non related tasks.

Soft parameter sharing is where each task has it's own model with it's own parameters where the parameters are encourage in some way to be related or similar. One example of this that we compare against are Cross-stitch networks[101], which model shared representations as a linear combination of input activation maps. NDDR[116] can generalize from

cross-stitch networks and utilize a very similar idea. Sluice networks[102] can generalize to a specific case of cross-stitch networks as well and are closely related to both cross-stitch and NDDR.

These types of soft parameter sharing methods require a full network architecture for each task, with separate parameters computing specific task-shared features between the two networks. This means they are much more computationally heavy then some of the smaller hard parameter sharing methods such as ours. They can also be harder to learn and harder to implement for many multi-task problems as they require training several single task networks and parameters to combine them and therefore are prone to over-fitting. Our method is much smaller and only slightly larger than a single task encoder-decoder architecture. A well-designed hard parameter sharing architecture for related tasks can outperform these other soft parameter sharing methods such as [101, 116, 102] as shown in Table 8.12 where our method outperforms all 3 of these methods on NYU40, while only requiring 0.017% more FLOPS than the single task network as opposed to more than double.

8.3 Results

We evaluate our method on several different datasets. Our ablations studies are done with the Scenenet dataset [14] because it is a synthetic dataset with many samples, clean depth/semantic images, and low noise/label error. The decision to evaluate on this data was done to avoid spurious errors caused by noise and labeling errors. For real world evaluation, we use both the NYUDv2 [15] and Scannetv2 [16] datasets to verify our results. For depth, we use the evaluation metrics from [158], which are the percent of pixels under log maximum relative depth error of $1.25^{\{1,2,3\}}$. For surface normals we use the same metrics as [141] which are the percent of pixels with angle error less than 11.25°, 22.5°, and 30° as well as mean angle error (MAE). For semantic labels, we use the mean class intersection over union score. Evaluations on each of these for a single modality/task baseline network

are shown in Table 8.1 for reference.

Table 8.1: Here are the individual results for each modality on the Scenenet dataset. For this, each method is trained and evaluated on only one task which is why we indicate - on the others. On each of these results, higher is better.

Single-Task	Normals $\% <$			Depth $\% <$			Semantics
Method	11.25	22.5	30	1.25	1.25^2	1.25^3	mIoU
Normals	83	91.5	93.8	-	-	-	-
Depth	-	-	-	86.1	95.3	97.5	-
Semantics	-	-	-	-	-	-	50.3

8.3.1 Surface Normal Ground Truth

To create ground truth surface normals, we use the method from [6] as it generates clean, semantically corrected surface normals. We use the parameters recommend by the authors, which are a depth in-painting window-size of 5, a normal max depth change factor of 0.02, a normal adaptive smoothing window of 10 for synthetic data and 30 for real-world data, and a planar threshold parameter of 0.4, which controls whether semantic planar surfaces are joined.

8.3.2 Architecture Hyper-parameters

We use the same training hyper-parameters to allow for fair evaluation. For the synthetic Scenenet ablation studies, we use a batch size of 16 320x240 images for 150,000 iterations with no data augmentation as Scenenet is considerably large and we wished to remove the effect data augmentation can have on training. For the real data, we use an input of 768x384 with a batch size of 8 for single tasks and 6 for multi-task with data augmentations of random flips, crops, and lighting changes used. We use a learning rate of 0.001 that has a polynomial decay with a decay step of 30,000 and a decay power of 0.9. We use a weight decay of 0.0005 and batch normalization decay of 0.99. We use an Adam solver with

$\beta_1 = 0.9$ and $\beta_1 = 0.999$. The eASPP parameters are the same as in [157] with an eAspp rate of 3, 6, and 12 for the three stages. For the four encoder stages, we use residual units of 3, 4, 6, and 3 and a filter size of 256, 512, 1024, and 2048 with strides of 1, 2, 2, and 1 respectively. We initialize our encoder with ImageNet weights in our experiments but for the newly learned layers we use He initialization [145] (see code [156]).

Table 8.2: Joint Normals and Semantics results with different fission methods when the loss is balanced ($\lambda_{\text{normals}} = 10.0$ and $\lambda_{\text{semantics}} = 1.0$) and auxiliary loses are used ($\alpha_1 = 0.6, \alpha_2 = 0.5$ for both tasks) with different initialization methods. Imp is the percent improvement metric and PIPFIP is percent improvement per flop increase percentage.

Method	Normals				Semantics	Imp	FLOPS	PIPFIP
	% < 11.25	% < 11.25	% < 30	MAE	mIoU	%	billion	%
Individually Trained Baselines								
Normals	83	91.5	93.8	8.3	-	-	38.43	-
Semantics	-	-	-	-	50.3	-	38.57	-
Trained From Scratch								
Early	84.9	92.7	94.9	7.6	51	1.84	56.56	0.04
Early-Mid	84.9	92.6	94.7	7.7	50.8	1.64	52.96	0.04
Mid	83.9	92.2	94.5	8	49.8	0.05	**38.58**	1.74
Late	85	92.6	94.8	7.6	50.9	1.8	38.64	9.92
Initialized from Labels								
Early	80.1	91.1	93.8	9.3	51	-1.05	56.56	0
Early-Mid	83.7	92.3	94.5	8	49.8	2.31	52.96	0.06
Mid	78.2	89.8	92.9	9.8	49	-4.2	**38.58**	0
Late	81.1	91.1	93.8	8.9	50.1	-1.34	38.64	0
Initialized from Normals								
Early	85.1	92.7	94.8	7.5	48.5	-0.52	56.56	0
Early-Mid	**86.6**	**93.5**	**95.4**	**6.9**	**51.6**	**3.58**	52.96	0.10
Mid	**86.6**	**93.4**	**95.3**	**7**	**51.3**	3.16	**38.58**	**121.99**
Late	85.6	92.9	95	7.3	50	1.27	38.64	7.00

8.3.3 Fission-scheme Ablation Study

To test the three proposed output fission schemes in Sections 8.2.3, 8.2.5, and 8.2.4, we do an ablation study on the Scenenet synthetic dataset shown in Table 8.2. We use the same encoder structure for all 4 decoder types with the two tasks chosen being surface normals and semantic labels with the architecture backbone and hyper parameters discussed in the Supplementary. We only used two tasks for this ablation study because we found the encoder starts to need more parameters as you add more tasks and we wanted to evaluate

against the normal Adapnet++ architecture from [157] without adding parameters in the encoder. Here we focus on semantic labels and surface normals. Results with depth and normals are shown in Table 8.3 and depth and semantic labels shown in 8.4.

For this ablation study, we used loss balancing of $\lambda_{\text{normals}} = 10$ and $\lambda_{\text{semantics}} = 1$ as this was found to be beneficial for all three methods in Section 8.3.6. We also use auxiliary losses for both surface normals and semantics with both using $\alpha_1 = 0.6$ and $\alpha_2 = 0.5$ as this was found to be beneficial in Section 8.3.5.We test all four output fission methods with three different initializations: a) the decoder trained from scratch and encoder initialized with ImageNet weights, b) initializing with the single task network trained on semantic labels, and c) initializing with the single task network trained on normals (both shown in Table 8.1). We tested with different initializations since many other methods such as [157] utilize this for multi-task learning.

Interestingly enough, when trained from scratch, all 4 methods outperform the single task method on surface normals, however, mid fission under-performs on semantic label prediction (compared to Table 8.1). Late fission is actually competitive with the standard method of early fission in these two tasks despite having far fewer task-specific parameters which implies that these tasks are either very interrelated (i.e. they have many efficient task-shared parameters) or that one of tasks only requires a few parameters. We can rule out the few parameter hypothesis given much literature on the number of parameters used for semantic label prediction as in [116] and the channel multiplier ablation study in the last Chapter in Table 7.8 for surface normals.

When initialized with the single task trained on semantic labels, the surface normal prediction quality suffers greatly and semantic label prediction also decreases for everything but early fission, which is counter-intuitive. Our theory is that a network trained on surface normals makes for a good initialization for multi-task learning; we see this throughout our ablation studies. When initializing with surface normals, surface normal prediction of all 4 methods greatly increases while semantic label prediction for early and late fission decrease

slightly, which is expected given we are emphasizing the surface normal loss. However, our proposed early-mid fission and mid fission schemes greatly increase semantic label prediction as well with this initialization due to task-shared features. It outperforms all other modes of early and late fission in both surface normal and semantic label prediction while also outperforming individual task predictions. Early-mid fission is almost as many FLOPS as early fission. However, the mid fission method only requires 68.3% of the FLOPS of early fission as shown in the bottom of Table 8.2 yet it outperforms early fission which is used by most current multi-task pixel-wise labeling methods. It might seem non-intuitive that mid fission is fewer FLOPS than late fission but this is due to the number of output channels of the final 1x1 convolutions. For example, 17x17 (14 semantic labels + 3 normals) requires more operations than 14x14 + 3x3. Due to mid fission's superior performance and minimum FLOPS, we select it for all further ablation studies.

Table 8.3: Joint Normals and Depth results using mid fission with loss balancing ($\lambda_{\text{normals}} = 10$ and $\lambda_{\text{depth}} = 1$). Here we compare when trained from 3 different initializations: Scratch being ImageNet fine-tuning, Normals being initialized with the network trained on normals, Depth being initialized with the network trained on depth. For this ablation, auxiliary loses are used ($\alpha_1 = 0.6, \alpha_2 = 0.5$ for both tasks).

Initialization	Normals % <			Depth % <		
	11.25	22.5	30	1.25	1.25^2	1.25^3
Scratch	66.4	86.2	90.7	80.8	89.4	92
Normals	**85.5**	**93.1**	**95.1**	81.8	90.9	94.1
Depth	74.5	88.4	91.9	**83.9**	**91.9**	**94.1**

In Table 8.3, we show results for mid fission for a model trained jointly on surface normals and depth with different initializations. Results for other fission methods were inferior as seen in Table 8.8. We initialize in three different ways: from scratch, from the normals network, and from the depth network. Note that for mid fission, as expected, initializing from normals improves normals substantially and improves depth some, while initializing from depth improves depth some and improves normals some. Given this, initializing from

normals still seems to be the better method. However, it is not as clear cut as the joint tasks of surface normals and semantic labels. In that case, both tasks are the highest when initialized with normals. Our hypothesis is that surface normals are a better initialization method since the model learns edge-like filters. It could also be that there are more task-shared features between surface normals and semantic labels as opposed to depth or that depth is just a harder task to learn.

Table 8.4: Joint Semantics and Depth results using mid fission with loss balancing ($\lambda_{semantics} = 2$ and $\lambda_{depth} = 1$). Here we compare when trained from 3 different initializations: Scratch being ImageNet fine-tuning, Normals being initialized with the network trained on normals, Depth being initialized with the network trained on depth. For this ablation, auxiliary loses are used ($\alpha_1 = 0.6, \alpha_2 = 0.5$ for both tasks).

Initialization	Depth $\% <$			Semantics
	1.25	1.25^2	1.25^3	mIoU
Scratch	33.5	60.8	76.7	**50.4**
Labels	40.8	65.2	80.1	**50.9**
Depth	77.2	91.4	95.5	43.1
Normals	**82.8**	**94.7**	**97.3**	**50.4**

In Table 8.4, we show results for mid fission for a model trained jointly on semantic labels and depth with different initializations. We initialize in three different ways just as in Table 8.3. Again, other fission methods were inferior as shown in Table 8.9 Note that for mid fission, as expected, initializing from labels improves labels slightly and improves depth some, while initializing from depth improves depth substantially but semantics drastically decreases accuracy. This again gives credence to our hypothesis that there are not as many task-shared features between depth and semantic labels. Interestingly enough, initializing from normals improves metrics for both even though it isn't a predicted task here. This confirms that initializing the network by training a method on surface normals creates good features for several related tasks. Given normals were seen as a very good task for transfer learning in Taskonomy [108], this makes some amount of sense.

8.3.4 Variability Ablation Study

To check how much our results would vary in Table 8.2, we ran a variability study by training our mid fission network on normals and semantics 3 different times to see if they converge to similar metrics. As shown in Table 8.5, we find our metrics only shift +/-0.1 at most verifying these results are meaningful and not just noise. Mid fission outperforms the single-task semantic labeling network by 1% mIOU, which is significantly more than the variance due to training.

Table 8.5: 3 Different training runs of our mid-fission network on normals and semantics with normal initialization and balanced losses.

Method	Normals				Semantics
	$\% < 11.25$	$\% < 11.25$	$\% < 30$	MAE	mIoU
Run 1	86.6	93.4	95.3	7	51.2
Run 2	86.6	93.5	95.3	7	51.4
Run 3	86.6	93.4	95.3	7	51.3
Range (+/-)	0.05	0.07	0.05	0.02	0.1

8.3.5 Auxiliary Loss Ablation Study

Recall Equations 8.1, 8.2, and 8.3 where each loss for the tasks has two auxiliary losses weighted with α_1 and α_2. Given that these are computed at the two stages of the decoder that are shared between tasks in mid fission, it is not clear whether these would help or cause task interference. In this study shown in Table 8.6, we aim to explore this issue. We evaluate the mid fission strategy without loss balancing using the same mid fission architecture and evaluation described in Section 8.3.3. Results considering both loss balancing and depth are shown in Tables 8.98.8.

To evaluate this, we test four different auxiliary loss strategies: no auxiliary losses denoted $\alpha = 0$ ($\alpha_1 = 0$, $\alpha_2 = 0$ for both normals and semantic labels), both auxiliary losses denoted $\alpha_* = 0.6, 0.5$ ($\alpha_1 = 0.6$, $\alpha_2 = 0.5$ for both semantics and normals), only

Table 8.6: Joint Normals and Semantics results with different aux losses using our proposed mid fission without loss balancing.

Method	Normals				Semantics
	$\% < 11.25$	$\% < 22.5$	$\% < 30$	MAE	mIoU
Individually Trained Baselines					
Normals	83	91.5	93.8	8.3	-
Semantics	-	-	-	-	50.3
Initialized from Labels					
All $\alpha = 0$	70.1	87.2	91.4	11.9	51.7
$\alpha_* = 0.6, 0.5$	71.3	87.5	91.6	11.6	51.6
$\alpha_s = 0$	76	89.1	92.5	10.4	52.2
$\alpha_n = 0$	61.9	84.4	89.8	13.6	51.7
Initialized from Normals					
All $\alpha = 0$	77.4	89.8	92.9	10.1	50.8
$\alpha_* = 0.6, 0.5$	79.3	90.5	93.5	9.4	51.8
$\alpha_s = 0$	81.9	91	94	8.7	50.4
$\alpha_n = 0$	75.8	89.5	92.9	10.4	51.7

auxiliary loss for normals denoted as $\alpha_s = 0$ ($\alpha_1 = 0.6$, $\alpha_2 = 0.5$ for normals, $\alpha_1 = 0$, $\alpha_2 = 0$ for semantic labels), and only auxiliary loss for semantic labels denoted as $\alpha_n = 0$ ($\alpha_1 = 0.6$, $\alpha_2 = 0.5$ for semantic labels, $\alpha_1 = 0$, $\alpha_2 = 0$ for normals). We do this for both semantic label initialization and surface normal initialization as this can have a joint effect on training.

Surprisingly, the semantic mean intersection over union (mIoU) is mostly unchanged when initialized from the single task trained solely on semantic labels, where the highest accuracy is actually where only surface normal auxiliary losses are used. The semantic prediction is even higher than our results in Table 8.2 but the surface normal prediction is significantly worse. When initialized from the single task trained solely on surface normals, surface normal metrics are much closer to the single task results whereas semantic label predictions are still outperforming the single task results and come close to results when

Table 8.7: Joint Normals and Semantics results with different loss balancing using our proposed mid fission.

Method	Normals				Semantics
	$\% < 11.25$	$\% < 22.5$	$\% < 30$	MAE	mIoU
Individually Trained Baselines					
Normals	83	91.5	93.8	8.3	-
Semantics	-	-	-	-	50.3
Initialized from Labels					
$\lambda_n = 1.0$	71.3	87.5	91.6	11.6	**51.6**
$\lambda_n = 5.0$	74.5	88.6	92.2	10.8	49.1
$\lambda_n = 10.0$	78.2	89.8	92.9	9.8	49
$\lambda_n = 15.0$	78.9	90.1	93.2	9.6	48.7
Initialized from Normals					
$\lambda_n = 1.0$	79.3	90.5	93.5	9.4	**51.8**
$\lambda_n = 5.0$	80.5	91	93.8	9.2	**51.8**
$\lambda_n = 10.0$	**86.6**	**93.4**	**95.3**	**7**	**51.2**
$\lambda_n = 15.0$	**86.6**	**93.4**	**95.3**	**7**	50

initialized by label prediction. Our hypothesis for this is that surface normals are a better initialization task given their reliance on edge and surface based features.

8.3.6 Loss Balancing Ablation Study

Empirical tests show that the semantic cross entropy loss in Equation 8.1 is approximately 10x the surface normal cosine loss from Equation 8.2 so we test loss balancing in Table 8.7. Again we evaluate the mid fission strategy using the same architecture and evaluation described in Section 8.3.3, with the auxiliary loss for both surface normals and semantic labels as described in Section 8.3.5. We evaluate with $\lambda_{\text{semantics}} = 1$ in all cases and $\lambda_{\text{normals}} = \{1, 5, 10, 15\}$ denoted as λ_n. Balancing with other modalities is shown in Table 8.8 and Table 8.9.

Unsurprisingly, when initializing with labels, increasing the λ_n of the cosine loss for

surface normals results in better surface normals but makes the semantic prediction underperform single-task prediction. However, when initializing with normals, semantic accuracy continues to outperform single task prediction (though decreasing slightly as λ_n increases), whereas surface normal prediction starts to outperform single task prediction. Note that at $\lambda_n = 15$, surface normal prediction remains the same but semantic labeling metrics decrease. Therefore, the choice of $\lambda_n = 10$ is validated here with greater than 4% more pixels being under 11.25 degrees error and almost 2% more mIoU compared to the single-task predictions.

Table 8.8: Joint Normals and Depth results with different fission methods both with no loss balancing and when the loss is balanced ($\lambda_{normals} = 10$ and $\lambda_{depth} = 1$). For this ablation, auxiliary loses are used ($\alpha_1 = 0.6, \alpha_2 = 0.5$ for both tasks).

Method	Normals % <			Depth % <		
	11.25	22.5	30	1.25	1.25^2	1.25^3
Baseline						
Normals	83	91.5	93.8	-	-	-
Depth	-	-	-	**86.1**	**95.3**	**97.5**
No Loss Balance						
Early	79.2	91.2	93	82.9	91.9	94.2
E-Mid	78.8	89.9	92.9	82.0	89.5	91.8
Mid	80.1	92	93.3	80.9	89.9	92.4
Late	76.6	89.2	92.4	81.6	89.1	91.4
$\lambda_n = 10$, $\lambda_d = 1$						
Early	83.6	92.2	94.5	81.1	91.5	94.3
E-Mid	85.4	92.8	94.8	83.6	92.6	94.8
Mid	**85.5**	**93.1**	**95.1**	81.8	90.9	93.2
Late	85	92.6	94.8	82.5	90.1	92.2

In Table 8.8, we show results for early, early-mid, mid, and late fission for a model trained jointly on surface normals and depth showing the impact of loss balancing. For early-mid and mid fission, we fine-tune from normals, otherwise we train from scratch as that doesn't help for early/late fission. Surface normal prediction for early, late, and

mid fission improve when using loss balancing, which makes sense given the depth loss is approximately 20x the normals loss. Interestingly enough, depth gets slightly better for both late and mid fission even though the depth loss has a lower overall impact on the total loss. This verifies the importance of loss balancing for mid fission regardless of the tasks learned. Note that early-mid and mid fission generally perform the best here.

Table 8.9: Joint Depth and semantic label results with different fission methods with no loss balancing and when the loss is balanced ($\lambda_{semantics} = 2$ and $\lambda_{depth} = 1$). For this ablation, auxiliary loses are used ($\alpha_1 = 0.6, \alpha_2 = 0.5$ for both tasks).

Initialization	Depth % <			Semantics
	1.25	1.25^2	1.25^3	mIoU
Baseline				
Depth	**86.1**	**95.3**	**97.5**	-
Semantics	-	-	-	50.3
No Loss Balance				
Early	39.2	64.8	79.1	48.5
E-Mid	73.4	90.6	95.2	49.4
Mid	80.1	93.1	96.5	49.2
Late	84.7	95.2	97.5	48.6
$\lambda_s = 2, \lambda_d = 1$				
Early	66	86.8	93.3	50.2
E-Mid	81.7	94.1	**97.1**	**51.2**
Mid	82.8	**94.7**	**97.3**	**51.0**
Late	82.3	**94.6**	**97.2**	48.7

In Table 8.9, we show results for early, early-mid, mid, and late fission for a model trained jointly on semantic labels and depth showing the impact of loss balancing. For early-mid and mid fission, we fine-tune from normals, otherwise we again train from scratch as we find that didn't help for early/late fission just as shown in Section 8.3.3. Semantic label prediction for all fission methods improve when using loss balancing, which makes sense given the depth loss is approximately 2x the semantic labels cross-entropy loss. Depth gets slightly better for both early, early-mid, and mid fission. This again shows

the importance of loss balancing for mid fission regardless of the tasks learned. Mid fission is always improved by balancing the losses. Early fission has very bad results for depth with these joint tasks. Our hypothesis is that there are not many good task-shared features between depth and semantic labels in the bottleneck generated by the encoder. Note that early-mid and mid fission generally perform the best here. This makes sense given further results in Table 8.4.

Balancing with all 3 modalities is shown in Table 8.10 but here we consider only surface normals and semantic labels as in the previous sections. Unsurprisingly, when initializing with labels, increasing the λ_n of the cosine loss for surface normals results in better surface normals but makes the semantic prediction under-perform single-task prediction. However, when initializing with normals, semantic accuracy continues to outperform single task prediction (though decreasing slightly as λ_n increases), whereas surface normal prediction starts to outperform single task prediction. Note that at $\lambda_n = 15$, surface normal prediction remains the same but semantic labeling metrics decrease. Therefore, the choice of $\lambda_n = 10$ is validated here with greater than 4% more pixels being under 11.25 degrees error and almost 2% more mIoU compared to the single-task predictions.

8.3.7 All 3 Modality Results

Using what was learned in the aforementioned ablation studies, we test the different fission methods with all 3 tasks, depth prediction, surface normal prediction, and semantic labeling.

In Table 8.10, we do the same ablation study as in Table 8.8 and Table 8.9 but now with all 3 tasks. We still use the same auxiliary loss strategy here. Note here that early, late, and mid fission all improve on normal prediction when balancing losses. However, late fission degrades on depth and semantic prediction tasks while early and early-mid fission degrade on depth prediction. Mid fission increases across the board as we have seen in our other ablation studies. The results are not as good as the two task results due to the difficulties of

Table 8.10: Joint 3 task results with different fission methods both with no loss balancing and when the loss is balanced ($\lambda_{semantics} = 1$, $\lambda_{depth} = 0.5$ and $\lambda_{normals} = 10$). E-Mid is Early-mid, N is the single task normals baseline, D the depth, and S the semantics.

Method	Normals % <			Depth % <			Semantics	Imp	FLOPS	PIPFIP
	11.25	22.5	30	1.25	1.25^2	1.25^3	mIoU	%	billion	%
Single Task										
Normals	83	91.5	93.8	-	-	-	-	-	38.43	0
Depth	-	-	-	86.1	95.3	97.5	-	-	38.40	0
Semantics	-	-	-	-	-	50.3	-	-	38.57	0
No Loss Balancing										
Early	76.2	89.5	92.7	80.2	89.1	92	49.7	-5.38	74.54	0
E-Mid	81.3	91.4	94.0	78.3	87.3	90.4	50.0	-3.89	67.35	0
Mid	70.8	86.5	90.8	77.1	87.5	90.2	49.0	-9.21	**38.58**	0
Late	73.3	88.1	91.9	81.8	90.7	93.4	**50.5**	-5.39	38.66	0
Balancing $\lambda_d = 0.5$,$\lambda_n = 10$										
Early	84.6	92.5	94.7	78.7	88.9	91.7	**50.6**	-1.99	74.54	0
E-Mid	**86.2**	**93.3**	**95.2**	77.2	89.1	92.9	**50.6**	-1.86	67.35	0
Mid	85	**93.1**	**95.1**	**84.8**	**95.3**	**97.5**	**50.4**	**0.86**	**38.58**	**33.32**
Late	84.2	92.4	94.6	78.9	90.8	93.7	49.4	-2.87	38.66	0

predicting 3 different tasks. This makes some amount of sense given depth is an absolute distance of a pixel from camera viewpoint and that is not necessarily a good indicator of the normal or label. Perhaps some far away objects are generally walls facing towards you but otherwise, by itself, this does not give a good representation. This is the reason we select surface normals and semantic labels as the two main tasks in other ablation studies and on our real dataset experiments. It's possible that different losses on the depth such as reverse huber, or relative depth, or tied losses between depth and normals could help improve both but that is left to future work.

Note that we also initialize the mid fission with the single task trained on surface normals but keep standard initialization for early and late fission as that was deemed best in Section 8.3.3. Loss balancing ($\lambda_{\text{depth}} = 0.5$, $\lambda_{\text{normals}} = 10$, $\lambda_{\text{semantics}} = 1$) is used in the last 3 columns and for all columns, all auxiliary losses are used. Note that while surface normal prediction and semantic labeling prediction are still outperforming the single task-

networks, depth prediction is equal in percent of pixels under 1.25^2 and 1.25^3 but slightly worse in percent of pixels under 1.25. Our hypothesis is that the number of encoder channels would have to be increased to generate more features relevant to the depth task; it must have more task-specific features and less task-shared features. We found this to be true in our ablation studies with our results using depth. The mid fission strategy still outperforms others even though early fission is widely used in the literature for multi-task prediction. Note that here mid fission only requires 51.9% of the FLOPS of early fission since adding a new task only adds a small number of new decoder parameters with one last block for each new task (0.017% FLOPS). Mid fission is also the only method with a positive $\%$ improvement and PIPFIP score.

Table 8.11: NYUDv2 13 class results with our proposed mid fission method compared to state of the art training on NYUDv2 13 class.

	Normals % $<$				Semantics
Method	11.25	22.5	30	mAE	mIoU
[93] from [115]	21.8	43.1	54.9	32.3	16.1
[115]	23.2	45.7	57.5	31.1	17.7
Ours	**50.1**	**70.8**	**78.1**	**20.6**	**42.1**

8.3.8 Real-world Results

We also test our proposed method on two real-world datasets, NYUDv2[15] and Scannet[16]. We chose to train and evaluate on the tasks of surface normals and semantic labels as they seem to have the greatest correlation and the most task-shared features as discovered by the ablation studies. Table 8.11 show multi-task results on the 13 class label subset. This dataset has very few training images so we initialize with our model with our network trained on Scannetv2. We also show results on the 40 class label subset of NYUDv2 in Table 8.12 with many other state-of-the-art multitask and single task methods. Note that some* train on normals from [89] quantized to RGB images. This creates several degrees

of normal error, resulting in $\% < 11.25$ being much worse but the rest of the metrics being better. We do not recommend this as it is not indicative of actual surface normals but show our method quantized as well to evaluate properly.

Table 8.12: NYUDv2 40 class results with our proposed mid fission method compared to state of the art (fine-tuned). - means the method does not train and evaluate on that task. * Indicates the normals are from [89] that were then quantized to RGB images.

	Normals % <				Semantics
Method	11.25	22.5	30	mAE	mIoU
Misra[101] from Gao et al.[116]*	48.6	76	86.5	15.2	34.8
Gao et al.[116]*	53.5	79.5	88.8	13.9	36.2
Ruder[102] from Gao et al.[116]*	49.7	77.1	88.0	14.8	34.9
Ours quantized*	40.9	82.7	90.5	15.2	38.5
Misra et al.[101]	39	54.4	60.2	34.1	19.3
Mousavian et al.[159]	-	-	-	-	39.2
Kokkinos et al.[111]	35.3	65.9	76.9	21.4	-
Xu et al.[106]	-	-	-	-	33.1
Hickson et al.[6]	59.5	72.2	77.3	19.7	-
Ours	**60.2**	**79.1**	**86.1**	**15.2**	**40.6**

Finally, we also train and evaluate on the Scannetv2 dataset in Table 8.13. We separate a subset of 10% of the scenes in the training set as validation set since there is no given validation split. We evaluated on this due to the large number of experiments done and GPU limitations for training. Many of the other methods such as [157] use different initialization to improve results. It is common to also crop sections of the test image with flips to evaluate several portions of the test image at full resolution in order to produce the best results. We avoided these techniques as they are not indicative of which architecture is most promising. Therefore, we evaluate by simply resizing the test image and running it through the network once. Other reported results are shown in the first half of the table for fairness followed by our trained results. These include our results for the backbone architecture [157], the standard early fission method, and our proposed mid fission method.

Late fission results weren't included as they were not as competitive in the previous ablation studies. Our proposed method is only slightly larger (0.017% more FLOPS) than the single-task architecture and 68.2% of the FLOPS of the standard early-fission method. We did not compare with methods that use several input sources (RGB+D,RGB+HHA) as we are evaluating on the single-to-many multi-task labeling domain.

Table 8.13: Scannetv2 validation set results. with our proposed mid fission method compared to state of the art. - means the method does not train and evaluate on that task.

	Normals % <				Semantics
Method	11.25	22.5	30	mAE	mIoU
[151]	-	-	-	-	27.5
Adapnet	-	-	-	-	47.3
DeepLabv3	-	-	-	-	50.1
[157]	-	-	-	-	50.3
[6]	50.1	63.2	68.2	28.8	-
Ours [157]	-	-	-	-	56.2
Ours Early	50.8	72.9	80.3	18.6	56
Ours Mid	**57.8**	**77.4**	**83.8**	**16.3**	**62.5**

Our proposed method considerably outperforms the competitive Adapnet++ baseline[157], which was the previous best result on RGB only input for Scannetv2. Figure 8.2 shows some qualitative results from the Scannetv2 validation set showing our impressive results.

8.4 Conclusion

In this chapter, we explore several different output fission methods for one-to-many multi-task pixel-wise prediction tasks. We show ablation studies on different fission methods and find that our proposed mid-fission method outperforms standard early fission methods with only 68% of the FLOPS (0.017% more FLOPS than single-task) for two tasks and only 51% for three tasks. We evaluate this on several different tasks on both synthetic and real datasets producing state of the art results of multiple modalities on multiple datasets. We

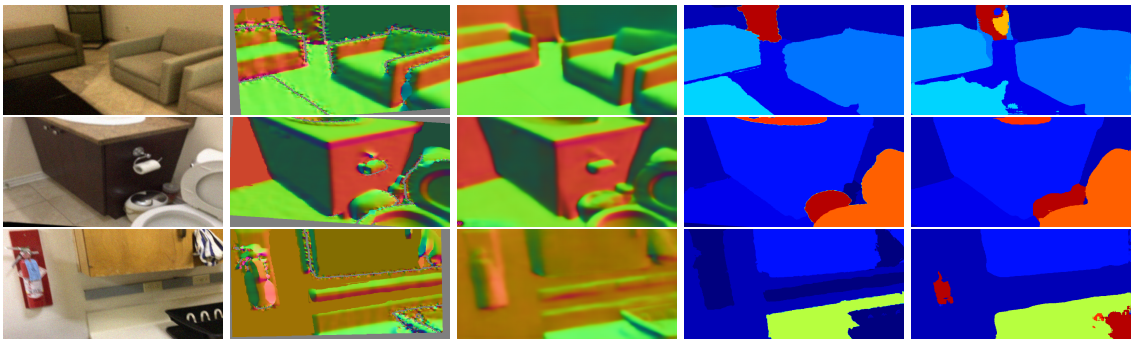


Figure 8.2: Illustration of the Scannetv2 data, together with the predictions of our model on them. Columns, from left to right: RGB image, ground truth surface normals, our normal predictions, ground truth semantic labels, then our semantic predictions.

release all of the code for reproducibility. We hope this can help other multi-task pixel-wise prediction methods achieve better results using this simple architecture change that can achieve higher accuracy with much fewer computations.

CHAPTER 9
CONCLUSIONS

In this dissertation, we discussed several approaches that utilize 3D contextual information to improve scene understanding. To reiterate the thesis statement:

Shape, used as an additional context, improves segmentation, unsupervised clustering, object classification and semantic labeling with little computational overhead. Combining shape and object labels improves results while (1) requiring few extra parameters, (2) provides better results using surface normals than depth, and (3) combining shape with labels improves accuracy for each task.

In Chapter 4, we proposed a hierarchical RGBD segmentation method for 3D videos. We demonstrated that combining color and depth cues improves unsupervised segmentation. We showed that a multistage segmentation with depth then color yields better results than a linear combination of depth and color. We also showed that using the depth in this manner improves segmentation results compared to just using color or naively combining color with depth.

In Chapter 5, we demonstrated a method to retrieve objects from 3D video sequences and learn to classify them into semantic categories without manual supervision. We showed that the dynamic 3D scene structure inherent to these videos can be leveraged to generate good object proposals. Our experiments on the challenging Cityscapes dataset showed meaningful retrieval of objects' proposals without supervision, and clustering into classes that visually correspond to common categories in scenes. We also showed how using 3D contextual scene information allows for good proposal discovery to learn useful clustering.

In Chapter 6, we also extract proposals as in Chapter 5 but instead learn supervised 3D shape and color features in order to semantically label 3D environments with a random

forest classifier as opposed to attempting to cluster proposals in an unsupervised manner with deep clustering. We also showed that training a classifier on a segmentation from a hierarchy of super pixels yields better results than training directly on super pixels, patches, or pixels as in previous work. Our method outperforms other methods that don't properly leverage 3D contextual information in the ways we do. We showed that surface normals yield a more meaningful representation that depth for this indoor semantic labeling task.

In Chapter 7, instead of using shape as another input signal, we attempt to learn a representation of shape from RGB images. To do this, instead of extracting proposals or segments like the previous chapters, we instead learn directly from pixels in a supervised manner using deep learning. We show our network can be constructed with few parameters so it works in real-time on a mobile device. We also demonstrate that jointly learning semantics and surface normals in one network actually improves results of both meaning that we can effectively learn good shared-task features between shape and semantics.

In Chapter 8, we extended the premise of Chapter 7 of jointly computing surface normals and semantic labels. This is different from the initial chapters where we used a shape representation as an intermediate step or input. This work further confirms that jointly combining surface normals with semantic labels improves results of both. We also showed that surface normals and other shape based encodings are better than raw depth for improving semantic labeling and classification when trained jointly. We also showed this could be done with few extra parameters with a multi-task architecture we call mid-fission (only 0.2% more FLOPS).

9.1 3D Contextual Information

In this dissertation, we focused on encoding and leveraging 3D contextual information to improve segmentation, object classification/retrieval, and semantic labeling. We will take a final note here to remind the reader what the context was in each of these chapters.

In Chapter 4, we used depth and camera calibration to construct 3D point clouds for segmentation. The contextual information here wasn't just the 3D scene itself, but also the knowledge that regions should never merge across depth boundaries despite color similarity. A person wearing a white t-shirt standing in front of a white wall should not be the same segment as the wall just because they are visually similar. Conversely, their face and pants and an object they are holding should not be the same segment just because they have a similar depth/position in the 3D environment. To encode this knowledge, we implement a two stage process that prevents color regions from merging across depth boundaries.

In Chapter 5, we used the segmentation pipeline from Chapter 4 to extract meaningful proposals from driving data. The contextual information here is the environment itself and it's constraints. The images come from a camera mounted on a car. We as humans know that the car is driving on roads and relevant objects are therefore things like people, cars, bicycles, and other things that are physically on top of the road. To encode this knowledge, we do ground plane estimation on the 3D environment to extract the road and sidewalks from the scene. We can then look for objects that are placed just on top of that ground plane with certain size constraints (we know the general 3D dimensions of cars, people, and bicycles).

In Chapter 6, we learn to semantically label 3D segments from hand-crafted features. The segments here are also retrieved by our method in Chapter 4 just like in the last chapter. However, here, we attempt to learn semantics from features which include the contextual information (recall from Figure 6.3: 3D position, shape of the segments, surface normals, and color histograms). Instead of manually encoding the context like the previous chapters, we attempt to learn them given these specific input features. The hope is that these methods can learn to leverage 3D information in ways we would not necessarily think of.

In Chapter 7, we learn to predict shape (in this case surface normals) and semantic labels from RGB images instead of using the 3D as an input like earlier chapters. In Chapter 6, we defined the context but tried to learn the encoding. Here we instead try to learn both

the context and encoding jointly. We do use contextual information to improve the ground truth surface normals used for learning. Given that surface normals can be very noisy, and we know how certain surfaces should look (i.e. floors are flat), we can correct the surface normals given this context.

In Chapter 8, we continue with the concepts proposed in the last chapter. Here instead of just focusing on predicting shape with semantics as a auxiliary task, we try to jointly predict shape and labels with few extra parameters. If you can learn to predict shape from RGB images and you can learn to predict labels from 3D colorized inputs, you should be able to predict both jointly from a single RGB input as the 3D information is already encoded as features. We constrain the problem further to attempt to learn both the context and encoding for both shape and labels with few task-independent features.

Appendices

APPENDIX A

SURFACE NORMAL TRAIN/TEST EVALUATION ACROSS DATASETS

Table A.1 shows an ablation study showing the results different datasets have in terms of generalization. Note that this isn't with our final method described in the paper. This is just an initial study on how different datasets affect the results and to determine what is the best way to mix the data.

Table A.1: Normal Accuracy comparisons with different testing and training datasets. The columns are the training set used and the rows are the evaluation accuracy for each individual dataset. *Scenenet NYU FT* means pretrained on Scenenet and finetuned on NYUDv2. *Scenenet+NYU* means the the dataset is trained with a batch-wise mix. *Scenenet+Scannet+NYU* means a batch-wise mix with all 3 datasets. *Average eval* is the evaluation for NYU, Scannet, and Scenenet averaged for each training split. The best result for each row is bold.

	Training set					
Accuracy	Scenenet	NYU	Scannet	Scenenet NYU FT	Scenenet+ NYU	Scenenet+ Scannet+NYU
NYU						
$\% < 11.25$	38.7	49.5	53.7	57	57.5	**59.5**
$\% < 22.5$	55.4	64.6	67.5	69.6	69.8	**72.2**
$\% < 30$	64.4	71.1	73.1	74.6	74.9	**77.3**
Mean Angle Error	27.5	23.6	22	21.3	21	**19.7**
Scannet						
$\% < 11.25$	20.2	29.2	49.3	36.7	36.7	**50.1**
< 22.5	43.7	47.2	**63.2**	54.6	54.9	**63.2**
< 30	56.2	54.9	**68.2**	60.9	62.1	**68.2**
Mean Angle Error	37.2	37.3	29	34.1	33.2	**28.8**
Scenenet						
$\% < 11.25$	65	12.2	16.4	21	**65.1**	64.5
$\% < 22.5$	70.9	33.3	44	48.2	**71**	70.7
$\% < 30$	72.9	45	57.2	59.9	**73**	68.2
Mean Angle Error	**25.9**	44.4	39.4	37.6	**25.9**	26.1
Average Eval						
$\% < 11.25$	41.30	30.30	39.80	38.2	53.10	**57.77**
$\% < 22.5$	56.67	48.37	58.23	57.5	65.23	**68.30**
$\% < 30$	64.50	57.00	66.17	65.1	70.00	**70.83**
Mean Angle Error	30.20	35.10	30.13	31	26.70	**25.00**

REFERENCES

[1] S. Hickson, S. Birchfield, I. Essa, and H. Christensen, "Efficient hierarchical graph-based segmentation of rgbd videos," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 344–351.

[2] S. Hickson, A. Angelova, I. Essa, and R. Sukthankar, "Unsupervised deep clustering for semantic object retrieval," in *Bay Area Machine Learning Symposium (BayLearn)*, http://www.baylearn.org/, 2017.

[3] ——, "Object category learning and retrieval with weak supervision," in *NeurIPS Learning with Limited Labeled Data Workshop*, IEEE, 2018, pp. 1068–1075.

[4] S. Hickson, A. Angelova, I. A. Essa, and R. Sukthankar, *Category learning neural networks*, US Patent App. 16/511,637, 2020.

[5] S. Hickson, I. Essa, and H. Christensen, "Semantic instance labeling leveraging hierarchical segmentation," in *2015 IEEE Winter Conference on Applications of Computer Vision*, IEEE, 2015, pp. 1068–1075.

[6] S. Hickson, K. Raveendran, A. Fathi, K. Murphy, and I. Essa, "Floors are flat: Leveraging semantics for real-time surface normal prediction," in *The IEEE International Conference on Computer Vision (ICCV) Workshops Geometry Meets Deep Learning (GMDL)*, 2019.

[7] B. Willimon, S. Hickson, I. Walker, and S. Birchfield, "An energy minimization approach to 3d non-rigid deformable surface estimation using rgbd data," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 2711–2717.

[8] D. Castro, S. Hickson, V. Bettadapura, E. Thomaz, G. Abowd, H. Christensen, and I. Essa, "Predicting daily activities from egocentric images using deep learning," in *proceedings of the 2015 ACM International symposium on Wearable Computers*, ACM, 2015, pp. 75–82.

[9] D. Castro, S. Hickson, P. Sangkloy, B. Mittal, S. Dai, J. Hays, and I. Essa, "Let's dance: Learning from online dance videos," *arXiv preprint arXiv:1801.07388*, 2018.

[10] S. Hickson, N. Dufour, A. Sud, V. Kwatra, and I. Essa, "Expressions in vr using eye tracking cameras," in *2017 Siggraph Emerging Technologies*, ACM, 2017.

[11] ——, "Eyemotion: Classifying facial expressions in vr using eye-tracking cameras," in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, IEEE, 2019, pp. 1626–1635.

[12] A. Sud, S. Hickson, V. Kwatra, and N. Dufour, *Classifying facial expressions using eye-tracking cameras*, US Patent App. 15/831,823, 2018.

[13] R. M. Haralick and L. G. Shapiro, "Image segmentation techniques," *Computer vision, graphics, and image processing*, vol. 29, no. 1, pp. 100–132, 1985.

[14] J. McCormac, A. Handa, S. Leutenegger, and A. J. Davison, "Scenenet rgb-d: Can 5m synthetic images beat generic imagenet pre-training on indoor segmentation," in *Proceedings of the International Conference on Computer Vision (ICCV)*, vol. 4, 2017.

[15] P. K. Nathan Silberman Derek Hoiem and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *ECCV*, 2012.

[16] A. Dai, A. X. Chang, M. Savva, M. Halber, T. A. Funkhouser, and M. Nießner, "Scannet: Richly-annotated 3d reconstructions of indoor scenes.," in *CVPR*, vol. 2, 2017, p. 10.

[17] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3213–3223.

[18] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *IJCV*, vol. 2, no. 59, pp. 167–181, 2004.

[19] M. Grundmann, V. Kwatra, M. Han, and I. Essa, "Efficient hierarchical graph-based video segmentation," in *CVPR*, 2010.

[20] C. Xu and J. J. Corso, "Evaluation of super-voxel methods for early video processing," in *CVPR*, 2012.

[21] S. Paris and F. Durand, "A topological approach to hierarchical segmentation using mean shift," in *CVPR*, 2007.

[22] J. Shi and J. Malik, "Normalized cuts and image segmentation," *TPAMI*, vol. 22, no. 22, pp. 888–905, 2000.

[23] E. Sharon, M. Galun, D. Sharon, R. Basri, and A. Brandt, "Hierarchy and adaptivity in segmenting visual scenes.," *Nature*, vol. 442, no. 7104, pp. 810–813, 2006.

[24] J. Papon, A. Abramov, M. Schoeler, and F. Worgotter, "Voxel cloud connectivity segmentation-supervoxels for point clouds," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2013, pp. 2027–2034.

[25] A. Abramov, K. Pauwels, J. Papon, F. Worgotter, and B. Dellen, "Depth-supported real-time video segmentation with the Kinect," in *IEEE Workshop on Applications of Computer Vision (WACV)*, IEEE, 2012, pp. 457–464.

[26] D. Weikersdorfer, A. Schick, and D. Cremers, "Depth-adaptive supervoxels for rgb-d video segmentation," in *ICIP*, 2013, pp. 2708–2712.

[27] D. Holz and S. Behnke, "Fast range image segmentation and smoothing using approximate surface reconstruction and region growing," in *Proceedings of the International Conference on Intelligent Autonomous Systems (IAS)*, 2012.

[28] A. Richtsfeld, T. Morwald, J. Prankl, J. Balzer, M. Zillich, and M. Vincze, "Towards scene understanding - object segmentation using rgbd-images," in *Computer Vision Winter Workshop (CVWW)*, 2012.

[29] J. Strom, A. Richardson, and E. Olson, "Graph-based segmentation for colored 3d laser point clouds," in *IROS, 2010*, IEEE, 2010, pp. 2131–2136.

[30] D. Munoz, J. A. Bagnell, N. Vandapel, and M. Hebert, "Contextual classification with functional max-margin markov networks," in *CVPR*, 2009.

[31] X. Ren, L. Bo, and D. Fox, "Rgb-(d) scene labeling: Features and algorithms," in *CVPR*, 2012.

[32] V. A. Prisacariu and I. D. Reid, "Pwp3d: Real-time segmentation and tracking of 3d objects," in *British Machine Vision Conference (BMVC)*, 2012.

[33] A. Teichman and S. Thrun, "Learning to segment and track in rgbd," in *The Tenth International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2012.

[34] H. Barlow, "Unsupervised learning," *Neural computation*, 1989.

[35] Y. Bengio, A. C. Courville, and P. Vincent, "Unsupervised feature learning and deep learning: A review and new perspectives," *CoRR, abs/1206.5538*, 2012.

[36] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *TPAMI*, 2013.

[37] Y. Bengio, "Deep learning of representations for unsupervised and transfer learning," in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, 2012, pp. 17–36.

[38] Y. Bengio *et al.*, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[39] P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol, "Extracting and composing robust features with denoising autoencoders," *ICML*, 2008.

[40] Q. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. Corrado, J. Dean, and A. Ng, "Building high-level features using large scale unsupervised learning," *ICML*, 2012.

[41] P. Agrawal, J. Carreira, and J. Malik, "Learning to see by moving," *CVPR*, 2015.

[42] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," *ICCV*, 2015.

[43] C. Doersch and A. Zisserman, "Multi-task self-supervised visual learning," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2051–2060.

[44] W. Lotter, G. Kreiman, and D. D. Cox, "Deep predictive coding networks for video prediction and unsupervised learning," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017.

[45] D. Pathak, R. Girshick, P. Dollar, T. Darrell, and B. Hariharan, "Learning features by watching objects move," *CVPR*, 2017.

[46] C. Vondrick, H Pirsiavash, and A. Torralba, "Anticipating visual representations from unlabeled video," *CVPR*, 2016.

[47] X. Wang and A. Gupta, "Unsupervised learning of visual representations using videos," *ICCV*, 2015.

[48] I. Misra, C. L. Zitnick, and M. Hebert, "Shuffle and learn: Unsupervised learning using temporal order verification," *ECCV*, 2016.

[49] M. Mirza, A. C. Courville, and Y. Bengio, "Generalizable features from unsupervised learning," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*, OpenReview.net, 2017.

[50] N. Srivastava, E. Mansimov, and R. Salakhudinov, "Unsupervised learning of video representations using lstms," in *International conference on machine learning*, 2015, pp. 843–852.

[51] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *Journal of Machine Learning Research*, vol. 11, no. Feb, pp. 625–660, 2010.

[52] S. Singh, A. Gupta, and A. A. Efros, "Unsupervised discovery of mid-level discriminative patches," *ECCV*, 2012.

[53] S. Kwak, M. Cho, I. Laptev, J. Ponce2, and C. Schmid, "Unsupervised object discovery and tracking in video collections," *ICCV*, 2015.

[54] J. Sivic, B. C. Russell, A. A. Efros, A. Zisserman, and W. T. Freeman, "Discovering objects and their location in images.," *ICCV*, 2005.

[55] B. C. Russell, A. A. Efros, J. Sivic, W. T. Freeman, and A. Zisserman, "Using multiple segmentations to discover objects and their extent in image collections," *CVPR*, 2006.

[56] S. Vijayanarasimhan, S. Ricco, C. Schmid, R. Sukthankar, and K. Fragkiadaki, "Sfm-net: Learning of structure and motion from video," *arXiv:1704.07804*, 2017.

[57] J. Sivic, B. C. Russell, A. Zisserman, W. T. Freeman, and A. A. Efros, "Unsupervised discovery of visual object class hierarchies," *CVPR*, 2008.

[58] E. Bart, I. Porteous, P. Perona, and M. Welling, "Unsupervised learning of visual taxonomies," *CVPR*, 2008.

[59] F. R. Bach and M. I. Jordan, "Learning spectral clustering," *NIPS*, 2005.

[60] F. Tian, B. Gao, Q. Cui, E. Chen, and T.-Y. Liu, "Learning deep representations for graph clustering," *In AAAI Conference on Artificial Intelligence*, 2014.

[61] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," *ICML*, 2016.

[62] B. Yang, X. Fu, N. Sidiropoulos, and M. Hong, "Towards k-means-friendly spaces: Simultaneous deep learning and clustering," *ICML*, 2017.

[63] J. Yang, D. Parikh, and D. Batra, "Joint unsupervised learning of deep representations and image clusters," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5147–5156.

[64] B. Wang, Z. Tu, and J. K. Tsotsos, "Dynamic label propagation for semi-supervised multi-class multi-label classification," *ICCV*, 2013.

[65] B. X. S. Jain and K. Grauman, "Fusionseg: Learning to combine motion and appearance for fully automatic segmentation of generic objects in video," *CVPR*, 2017.

[66] C. S. Pavel Tokmakov Karteek Alahari, "Learning video object segmentation with visual memory," *ICCV*, 2017.

[67] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[68] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CVPR*, 2015.

[69] K. Simonyan and A. Zisserman, "Networks for large-scale image recognition," *ICLR*, 2015.

[70] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CVPR*, 2016.

[71] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *NIPS*, 2015.

[72] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik, "Learning rich features from rgb-d images for object detection and segmentation," in *European Conference on Computer Vision*, Springer, 2014.

[73] O. M. Mozos, R. Triebel, P. Jensfelt, A. Rottmann, and W. Burgard, "Supervised semantic labeling of places using information extracted from sensor data," *Robotics and Autonomous Systems*, vol. 55, no. 5, pp. 391–402, 2007.

[74] A. Nüchter and J. Hertzberg, "Towards semantic maps for mobile robots," *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 915–926, 2008.

[75] R. B. Rusu, "Semantic 3d object maps for everyday manipulation in human living environments," *KI-Künstliche Intelligenz*, vol. 24, no. 4, pp. 345–348, 2010.

[76] H. S. Koppula, A. Anand, T. Joachims, and A. Saxena, "Semantic labeling of 3d point clouds for indoor scenes," in *Advances in neural information processing systems*, 2011, pp. 244–252.

[77] S. Gupta, P. Arbelaez, and J. Malik, "Perceptual organization and recognition of indoor scenes from rgb-d images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 564–571.

[78] C. Couprie, C. Farabet, L. Najman, and Y. LeCun, "Indoor semantic segmentation using depth information: 1st international conference on learning represen-

tations, iclr 2013," in *1st International Conference on Learning Representations, ICLR 2013*, 2013.

[79] A. Wang, J. Lu, G. Wang, J. Cai, and T.-J. Cham, "Multi-modal unsupervised feature learning for rgb-d scene labeling," in *European Conference on Computer Vision*, Springer, 2014, pp. 453–467.

[80] D. Lin, S. Fidler, and R. Urtasun, "Holistic scene understanding for 3d object detection with rgbd cameras," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 1417–1424.

[81] J. Carreira and C. Sminchisescu, "Cpmc: Automatic object segmentation using constrained parametric min-cuts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 7, pp. 1312–1328, 2012.

[82] J. Stückler, B. Waldvogel, H. Schulz, and S. Behnke, "Dense real-time mapping of object-class semantics from rgb-d video," *Journal of Real-Time Image Processing*, vol. 10, no. 4, pp. 599–609, 2015.

[83] A. Bansal, B. Russell, and A. Gupta, "Marr revisited: 2d-3d alignment via surface normal prediction," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 5965–5974.

[84] D. Hoiem, A. A. Efros, and M. Hebert, "Recovering surface layout from an image," *International Journal of Computer Vision*, vol. 75, no. 1, pp. 151–172, 2007.

[85] A. G. Schwing, S. Fidler, M. Pollefeys, and R. Urtasun, "Box in the box: Joint 3d layout and object reasoning from single images," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 353–360.

[86] L. Ladicky, B. Zeisl, and M. Pollefeys, "Discriminatively trained dense surface normal estimation," in *ECCV*, 2014.

[87] D. F. Fouhey, A. Gupta, and M. Hebert, "Data-driven 3d primitives for single image understanding," in *2013 IEEE International Conference on Computer Vision*, 2013, pp. 3392–3399.

[88] X. Wang, D. Fouhey, and A. Gupta, "Designing deep networks for surface normal estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 539–547.

[89] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *The IEEE International Conference on Computer Vision (ICCV)*, 2015.

[90] P. Wang, X. Shen, B. Russell, S. Cohen, B. Price, and A. L. Yuille, "Surge: Surface regularized geometry estimation from a single image," in *Advances in Neural Information Processing Systems*, 2016, pp. 172–180.

[91] C. Liu, K. Kim, J. Gu, Y. Furukawa, and J. Kautz, "Planercnn: 3d plane detection and reconstruction from a single image," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4450–4459.

[92] Y. Zhang, S. Song, E. Yumer, M. Savva, J.-Y. Lee, H. Jin, and T. Funkhouser, "Physically-based rendering for indoor scene understanding using convolutional neural networks," in *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, IEEE, 2017, pp. 5057–5065.

[93] Z. Ren and Y. J. Lee, "Cross-domain self-supervised multi-task feature learning using synthetic imagery," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[94] C. Godard, O. Mac Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 270–279.

[95] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International journal of computer vision*, vol. 47, no. 1-3, pp. 7–42, 2002.

[96] F. Liu, C. Shen, and G. Lin, "Deep convolutional neural fields for depth estimation from a single image," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 5162–5170.

[97] T. Zhou, M. Brown, N. Snavely, and D. Lowe, "Unsupervised learning of depth and ego-motion from video," *CVPR*, 2017.

[98] R. Garg, V. K. BG, G. Carneiro, and I. Reid, "Unsupervised cnn for single view depth estimation: Geometry to the rescue," in *European Conference on Computer Vision*, Springer, 2016, pp. 740–756.

[99] R. Caruana, "Multitask learning: A knowledge-based source of inductive bias," in *Proceedings of the Tenth International Conference on Machine Learning*, Morgan Kaufmann, 1993, pp. 41–48.

[100] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niebner, M. Savva, S. Song, A. Zeng, and Y. Zhang, "Matterport3d: Learning from rgb-d data in indoor environments," in *2017 International Conference on 3D Vision (3DV)*, IEEE, 2017, pp. 667–676.

[101] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, "Cross-stitch networks for multi-task learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3994–4003.

[102] S. Ruder, J. Bingel, I. Augenstein, and A. Søgaard, "Learning what to share between loosely related tasks," *arXiv preprint arXiv:1705.08142*, 2017.

[103] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[104] O. H. Jafari, O. Groth, A. Kirillov, M. Y. Yang, and C. Rother, "Analyzing modular cnn architectures for joint depth prediction and semantic segmentation," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 4620–4627.

[105] R. Kuga, A. Kanezaki, M. Samejima, Y. Sugano, and Y. Matsushita, "Multi-task learning using multi-modal encoder-decoder networks with shared skip connections," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 403–411.

[106] D. Xu, W. Ouyang, X. Wang, and N. Sebe, "Pad-net: Multi-tasks guided prediction-and-distillation network for simultaneous depth estimation and scene parsing," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 675–684.

[107] K.-K. Maninis, I. Radosavovic, and I. Kokkinos, "Attentive single-tasking of multiple tasks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[108] A. R. Zamir, A. Sax, W. Shen, L. J. Guibas, J. Malik, and S. Savarese, "Taskonomy: Disentangling task transfer learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3712–3722.

[109] T. Dharmasiri, A. Spek, and T. Drummond, "Joint prediction of depths, normals and surface curvature from rgb images using cnns," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, IEEE, 2017, pp. 1505–1512.

[110] V. Nekrasov, T. Dharmasiri, A. Spek, T. Drummond, C. Shen, and I. Reid, "Real-time joint semantic segmentation and depth estimation using asymmetric annotations," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 7101–7107.

[111] I. Kokkinos, "Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory.," in *CVPR*, vol. 2, 2017, p. 8.

[112] X. Qi, R. Liao, Z. Liu, R. Urtasun, and J. Jia, "Geonet: Geometric neural network for joint depth and surface normal estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 283–291.

[113] Z. Yang, P. Wang, W. Xu, L. Zhao, and R. Nevatia, "Unsupervised learning of geometry with edge-aware depth-normal consistency," *arXiv preprint arXiv:1711.03665*, 2017.

[114] A. Kendall, Y. Gal, and R. Cipolla, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7482–7491.

[115] S. Liu, E. Johns, and A. J. Davison, "End-to-end multi-task learning with attention," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[116] Y. Gao, J. Ma, M. Zhao, W. Liu, and A. L. Yuille, "Nddr-cnn: Layerwise feature fusing in multi-task cnns by neural discriminative dimensionality reduction," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[117] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *IROS, 2012*, IEEE, 2012, pp. 573–580.

[118] *Efficient hierarchical graph-based segmentation of rgbd videos project page*, `https://www.cc.gatech.edu/cpl/projects/4dseg/`.

[119] J. Barron and N. A Thacker, "Tutorial: Computing 2d and 3d optical flow.," Imaging Science and Biomedical Engineering Division, Medical School, University of Manchester, Tech. Rep., 2005.

[120] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Image analysis*, Springer, 2003, pp. 363–370.

[121] M. Livingstone and D. Hubel, "Segregation of form, color, movement, and depth: Anatomy, physiology, and perception," *Science*, vol. 240, no. 4853, pp. 740–749, 1988.

[122] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.

[123] H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H. C. Wolf, "Parametric correspondence and chamfer matching: Two new techniques for image matching.," *IJCAI*, pp. 659–663, 1977.

[124] C. Xu, C. Xiong, and J. J. Corso, "Streaming hierarchical video segmentation," in *ECCV*, 2012.

[125] S. Grossberg, "3-d vision and figure-ground separation by visual cortex," *Perception and Psychophysics*, 1994.

[126] B. Alexe, T. Deselares, and V. Ferrari, "Measuring the objectness of image windows," *PAMI*, vol. 34, no. 11, 2012.

[127] J. Uijlings, K. van de Sande, and T. Gevers, "Selective search for object recognition," *IJCV*, 2013.

[128] S. Holzer, R. B. Rusu, M. Dixon, S. Gedikli, and N. Navab, "Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, IEEE, 2012, pp. 2684–2689.

[129] L. Bottou and Y. Bengio, "Convergence properties of the k-means algorithms," in *Advances in neural information processing systems*, 1995, pp. 585–592.

[130] M. Lin, Q. Chen, and S. Yan, "Network in network," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2014.

[131] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, IEEE, vol. 2, 2006, pp. 1735–1742.

[132] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *CVPR*, 2016.

[133] *Robot demo of semantic labeling leveraging hierarchical segmentation*, `https://www.youtube.com/watch?v=oZeHPolLy7M`.

[134] O. Martinez Mozos, A. Rottmann, R. Triebel, P. Jensfelt, W. Burgard, *et al.*, "Semantic labeling of places using information extracted from laser and vision sensor data," *From Sensors to Human Spatial Concepts*, 2006.

[135] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *Robotics and automation (ICRA), 2011 IEEE International Conference on*, IEEE, 2011, pp. 1–4.

[136] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[137] D. G. Lowe *et al.*, "Object recognition from local scale-invariant features.," in *ICCV*, vol. 99, 1999, pp. 1150–1157.

[138] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual categorization with bags of keypoints," in *Workshop on statistical learning in computer vision, ECCV*, Prague, vol. 1, 2004, pp. 1–2.

[139] C Cadena Lerma and J Kosecka, "Semantic parsing for priming object detection in rgb-d scenes," IEEE, Workshop on Semantic Perception, Mapping and Exploration (SPME) in conjuction with ICRA, 2013, 2013, pp. 1–6.

[140] *Video for floors are flat: Leveraging semantics for real-time surface normal prediction*, `https://www.youtube.com/watch?v=QrXqmUBlmbc`.

[141] D. F. Fouhey, A. Gupta, and M. Hebert, "Data-driven 3d primitives for single image understanding," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 3392–3399.

[142] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[143] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[144] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, Springer, 2015, pp. 234–241.

[145] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[146] Google, *Tensorflow lite*, `https://www.tensorflow.org/lite/`, Accessed: 2018-11-14, 2018.

[147] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Networks for Machine Learning*, vol. 4, no. 2, 2012.

[148] Q. Geng, Z. Zhou, and X. Cao, "Survey of recent progress in semantic image segmentation with cnns," *Science China Information Sciences*, vol. 61, no. 5, p. 051 101, 2018.

[149] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, IEEE, 2014, pp. 3431–3440.

[150] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1520–1528.

[151] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[152] S. Ruder, "An overview of multi-task learning in deep neural networks," *arXiv preprint arXiv:1706.05098*, 2017.

[153] C. Cadena and J. Košecká, "Semantic segmentation with heterogeneous sensor coverages," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2014, pp. 2639–2645.

[154] R. Zhang, S. A. Candra, K. Vetter, and A. Zakhor, "Sensor fusion for semantic segmentation of urban scenes," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 1850–1857.

[155] C. Hazirbas, L. Ma, C. Domokos, and D. Cremers, "Fusenet: Incorporating depth into semantic segmentation via fusion-based cnn architecture," in *Asian conference on computer vision*, Springer, 2016, pp. 213–228.

[156] *Code for sharing decoders: An exploration of network fission for multi-task pixel labeling*, `https://github.com/StevenHickson/Adapnet-Shape`.

[157] A. Valada, R. Mohan, and W. Burgard, "Self-supervised model adaptation for multimodal semantic segmentation," *International Journal of Computer Vision*, pp. 1–47,

[158] L. Ladicky, J. Shi, and M. Pollefeys, "Pulling things out of perspective," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 89–96.

[159] A. Mousavian, H. Pirsiavash, and J. Košecká, "Joint semantic segmentation and depth estimation with deep convolutional networks," in *2016 Fourth International Conference on 3D Vision (3DV)*, IEEE, 2016, pp. 611–619.